Civil Engineer

INVESTIGATIONS OF PRIMAL TECHNIQUES FOR INTEGER LINEAR PROGRAMS

by

MARK ANDREW HOPPER

Course I 17 May 1968

INVESTIGATIONS OF PRIMAL TECHNIQUES FOR INTEGER LINEAR PROGRAMS

BY

MARK ANDREW HOPPER
SB, United States Naval Academy
(1965)

Submitted in partial fulfillment

of the requirements for the degree of

Civil Engineer

at the

Massachusetts Institute of Technology

(1968)

ABSTRACT

INVESTIGATIONS OF PRIMAL TECHNIQUES FOR INTEGER LINEAR PROGRAMS

by
MARK ANDREW HOPPER

Submitted to the Department of Civil Engineering on 17 May 1968 in partial fulfillment of the requirements for the degree of Civil Engineer.

A primal feasible (all-integer) integer linear programming algorithm, developed independently by R. H. Gonzales and R. D. Young, has been coded for a digital computer and has been implemented as a problem-oriented language under the ICES/OPTECH subsystem for mathematical programming. Complete program documentation and a guide for the use of this system are included.

Some convergence problems of the Gonzales-Young algorithm are identified and geometric interpretations that help to illuminate these problems are offered.

A strong relationship between the Gonzales-Young algorithm and R. E. Gomory's Algorithm I (a dual feasible method) is established. This relationship and the interpretation of convergence problems in the Gonzales-Young algorithm are combined to produce a modified, and hopefully more efficient, primal algorithm. Performance of this algorithm is measured against other integer linear programming algorithms on a small set of test problems.

The concepts of the modified primal algorithm for the all-integer problem are enlarged upon to develop the essentials of the first primal mixed integer-continuous linear programming algorithm that has ever been proposed. No computational results of this algorithm are available.

Thesis Advisor: Dr. Alan M. Hershdorfer

Title: Assistant Professor of Civil Engineering

ACKNOWLEDGMENTS

I would like to thank Professor Alan M. Hershdorfer, my thesis advisor, for the patience and assistance he has offered during the course of this research.

I am also indebted to Professor Felipe Ochoa-Rosso, an exemplary teacher, who first introduced me to the subject matter of this topic and who offered encouragement and suggestions during the initial phases of this work.

I would also like to thank Mr. David Bivins for his assistance with regard to the computer programming phase of this work; in particular, his information about the OPTECH subsystem.

I particularly want to thank the Civil Engineer Corps of the United States Navy for allowing me to interrupt my military duties in order to continue my education. The United States Navy has provided all of the financial support connected with this effort.

Finally, I would like to express my gratitude to my wife, Jeannie, who provided reassuring support and constant optimism during the course of this work and who did an excellent job of typing the manuscript under some very trying circumstances.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTORY REMARKS

1.1 Integer Linear Programming Problems

The formulation of an Integer Linear Programming (ILP) problem is similar to that of the more widely known Linear Programming (LP) problem. Like the LP formulation the ILP formulation is characterized by a linear objective function that is to be optimized, subject to a set of linear convexity constraints and non-negativity restrictions on the problem variables. However, the ILP formulation imposes the additional restriction that some or all of the problem variables must take on integer values in a feasible solution.

The ILP formulation arises quite frequently in both business and engineering. Integrality restrictions on the problem variables sometimes result from the inherent indivisibility of the units that these variables represent, such as men and machines. At other times integrality is employed in the formulation in a somewhat artificial manner since it can be used conveniently to express logical relationships (e.g. decision variables). Regardless of how the integrality restrictions arise, special procedures are necessary to ensure that they are satisfied.

1.2 Historical Approaches to the Problems

Since the ILP formulation is so similar to the LP formulation, it

was a natural consequence that the first attempts to solve the ILP problem centered around the already proven Simplex method for linear programming. The first such approach was forwarded by G. B. Dantzig [1]. This technique attacked the All Integer Linear Programming (AILP) problem by successively modifying the linear program that was inherent in the AILP formulation (if the integrality restrictions were ignored). The key aspect of Dantzig's approach was that he deduced new, more binding, constraints from the optimal tableau of the linear program. These additional constraints did not exclude any feasible integer points, but they made the available non-integer optimal extreme point infeasible. Thus the addition of such constraints to the previous constraint set resulted in a reduction of the solution space and re-optimization was required. The supposition was that by successively reducing the solution space, while not excluding integer solutions, eventually one of the reoptimized tableaus in the sequence would have to display the desired integrality properties in its solution [2]. Although counterexamples ultimately showed that Dantzig's technique could not guarantee a finite convergence, the concept helped to launch the very important class of solution techniques for ILP problems that are collectively referred to as the "cutting plane" methods. The commonality among these techniques arises from the fact that they all employ additional constraints (cuts) that function in a fashion similar to those proposed by Dantzig's approach.

The first proven algorithms for solving ILP problems by the cutting plane approach were developed by R. E. Gomory [3,4,5]. The cuts that

are generated in these algorithms are founded on a much more sophisticated mathematical basis than the proposals of Dantzig. In fact, while Dantzig can be credited with proposing the first legitimate cut, Gomory's work continues to provide the basis for most of the meaningful developments within the cutting plane context. Gomory's first algorithms have been followed by numerous investigations into the nature of the various cuts that can be generated in his algorithms. The manner in which cuts are chosen from the many that are available at a given stage of the algorithms has been shown to affect the performance of the various algorithms to a considerable degree [6].

1.3 The Motivation for a Primal Algorithm

Even today there is no single algorithm that can be said to handle all ILP problems in a satisfactory manner. This is evidenced by the fact that the topic continues to enjoy a good deal of attention in technical publications. Gomory's algorithms and the various modifications to them have enjoyed only sporadic success. Some problems have been solved in what has been considered a reasonable amount of time while others have proved computationally expensive beyond tolerable limits. Unhappily the computation required for the solution process has not always been found to be proportional to the size of the problem, so that no reliable guidelines are available for predicting the cost of solving a particular problem. This difficulty has been compounded by the fact that, until recently, all of the cutting plane algorithms have

been developed as "dual feasible" methods. This characteristic meant that no primal feasible solution to the ILP was available until the optimal solution was found. As a consequence all of the problems that proved too costly to solve to optimality resulted in a total waste of computational effort. Thus the need for a "primal feasible" method was evident. Such a method would provide at least a feasible solution to the problem under consideration. This solution, though not proven to be optimal, could be useful and sometimes might be fairly close to the optimal.

1.4 Contributions to the Primal Algorithm

In June of 1965 Gonzales [7] and Young [8] reported independent development of the first primal feasible algorithms designed to solve the AILP problem. The two algorithms are basically the same in that they are both direct extensions of proposals made by Ben-Israel and Charnes [9]. The only substantial difference between the two algorithms lies in the tableau representation and not in the computational approach. Both of these algorithms use a cut that was developed by Gomory in what is referred to as his Algorithm II [6]. In fact, at least in their basic forms, these algorithms represent a modification of Gomory's Algorithm II whereby primal pivot selection rules are substituted for dual pivot selection rules. This does not in any way detract from the importance of either Young's or Gonzales' work. Since Gomory did not address himself to the applicability of his cuts to a primal approach

the credit for the primal algorithm belongs to Young and Gonzales. Young showed that his version of the algorithm could be made to converge in a finite number of iterations. Although Gonzales was not able to prove finite convergence save for the two dimensional problem, he did provide a clever method for achieving a starting basic feasible solution when one is not available in the initial tableau. In order to give dual credit for the development of the primal algorithm, the two versions will be referred to here as one and will be called the Gonzales-Young (G-Y) algorithm [10].

1.5 Scope of the Current Study

The work that is reported in this paper was carried on in two related parts. The primary objective of this research has been to provide for ICES/OPTECH Subsystem [11] a useable capability for solving AILP problems in the form of the Gonzales-Young algorithm. This capability will not only be valuable to researchers in this particular branch of mathematical programming, but will also provide a useful tool for research in civil engineering and management in general.

At the same time that work was being carried forward in the computer implementation phase, the mechanics of the G-Y algorithm were being investigated with a view toward improving the convergence characteristics of the algorithm itself. Originally this second aspect of the research was intended to uncover an optimum pivot selection criteria and perhaps to discover ways of generating cuts that would be more binding than the ones that are generated in the G-Y algorithm. In this regard the author was able to find some primal cuts

that were better than the normal G-Y cuts. Unfortunately each of these new cuts resulted in a loss of the all-integer tableau. As a result these cuts could not be used in the G-Y context of all integer solutions. However, the demonstrated power of the different cuts led the author to attempt a modification of the G-Y algorithm. The motivations behind this modification and the results that have been attained are reported in the second part of this paper. Finally, the basic approach of this modified AILP algorithm is extended to produce a primal algorithm for the MILP problem.

To summarize, the research reported in this paper has been presented in two parts. The first part describes in detail the considerations that went into the computer implementation of the G-Y algorithm (called OPAILP). This part has been written in such a way that it can stand alone as a user's guide for the OPTECH all integer linear programming capability. The second distinct part includes an interpretation of some of the convergence problems of the G-Y algorithm that served to motivate the proposal of a modified AILP algorithm. This discussion is extended to relate the G-Y algorithm to Gomory's Algorithm I for the AILP problem. A modified algorithm is proposed that is essentially a hybred of the G-Y algorithm and Algorithm I. Like the G-Y algorithm the modified algorithm is primal feasible, but it specifically avoids the difficulties that have been shown to impede the G-Y convergence process. The performance of this hybred algorithm is measured against several AILP algorithms (including G-Y) on several problems. Finally, an outline of the essentials of a primal MILP algorithm are given, though no computational results are reported.

CHAPTER I FOOTNOTES

[1] Dantzig, G. B., "Discrete-Variable Extremum Problems", Operations Research, Vol. 5 (1957), pp. 266-277.

[2] Actually Dantzig did not argue in exactly this way, but this is the point of his algorithm.

[3] Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs", Princeton-IBM Mathematics Research Project, Technical Report Number 1, November 17, 1958.

[4] Gomory, R. E., "All-integer Integer Programming Algorithm", IBM Research Center, Research Report RC-189, January 29, 1960.

[5] Gomory, R. E., "An Algorithm for the Mixed Integer Problem", RM-2597 Rand Corporation, July 7, 1960.

[6] Balinski, M. L., "Integer Programming: Methods, Uses, Computation", Management Science, Vol. 12, No. 3, November, 1965, pp. 253-309.

[7] Gonzales-Zubieta, R. H., On Some Aspects of Integer Linear Programming, Technical Report No. 16, Operations Research Center, Massachusetts Institute of Technology, June, 1965.

[8] Young, R. D., "A Primal (all-integer) Integer Programming Algorithm", Journal of Research of the National Bureau of Standards-B, Mathematics and Mathematical Physics, Vol. 69B, No. 3, July-September, 1965, pp. 213-250.

[9] Ben-Israel and Charnes, A., "On Some Problems of Diophantine Programming", Cahiers de Centre d'Etudes de Recherche Operationelle, pp. 215-280, 1962.

[10] Precedent for referring to the algorithm by this name has already been set by: Ochoa-Rosso, F., et.al., OPTECH User's Manual, Department of Civil Engineering, Massachusetts Institute of Technology, October 1967.

[11] IBID.

CHAPTER II

THE GONZALES-YOUNG AILP ALGORITHM

In order to provide a reasonable basis for subsequent discussions of the computer implementation of the G-Y algorithm, this chapter will outline the fundamental steps that the algorithm performs. The problem formulation and notation employed here were used by Gomory in his first ILP algorithm and by Gonzales in his presentation of the primal AILP algorithm. This approach was chosen over that of Young because it facilitates comparisons among the ILP algorithms. These comparisons are undertaken in the second part of this paper, mainly in Chapter IV.

2.1 The Problem: Notation and Formulation

The G-Y primal AILP algorithm will solve a problem that can be formulated in the following way:

Find non-negative integers x_j $j=1,\ldots,n$

that maximize

$$\sum_{j=1}^{n} c_j x_j \qquad (2.1.1)$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i=1,\ldots m \qquad (2.1.2)$$

with the c_j, a_{ij}, and b_i integers

Except for the integrality restrictions on the problem variables this formulation is exactly that of the classical LP formulation. This problem notation is altered slightly by Gonzales in order to recast the problem into a parametric form suggested by A. W. Tucker.
Each of the inequality constraints (2.1.2) is made into an equation by adding a slack variable s_i. Since all of the constraint coefficients are integer and the x_j are restricted to non-negative integers, the s_i are also restricted to non-negative integer values for a feasible solution. Non-negatively constrained parametric variables t_j are introduced to act as proxies for the original problem variables. For notational consistency the c_j are referred to as $a_{0,j}$ and the b_i as $a_{i,0}$. Thus in terms of the Tucker notation the problem can be restated in the following form:

maximize z

when:
$$z=a_{0,0} + \sum_{j=1}^{n} -a_{0,j}(-t_j) \qquad (2.1.3)$$

$$s_i=a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j) \quad i=1,\ldots,m \qquad (2.1.4)$$

$$x_j= \quad (-1)(-t_j) \quad j=1,\ldots,n \qquad (2.1.5)$$

where the t_j are restricted to non-negative integer values and z is an unsigned variable.

The above relations can be expressed in a tableau format as follows:

	$-t_1$			$-t_n$
$z =$	$a_{0,0}$	$a_{0,1}$		$a_{0,n}$
$s_1 =$	$a_{1,0}$	$a_{1,1}$		$a_{1,n}$
.				
.				
$s_m =$	$a_{m,0}$	$a_{m,1}$		$a_{m,n}$
$x_1 =$	$a_{m+1,0}$	$a_{m+1,1}$		$a_{m+1,n}$
.				
.				
$x_n =$	$a_{m+n,0}$	$a_{m+n,1}$		$a_{m+n,n}$

Figure 2.1.1

In this tableau the problem variables are listed along the left margin. The solution that corresponds to any given tableau is found in the zeroth column. The problem variables may be viewed as slack variables associated with the hyperplane representation of the problem constraints in non-basic (parametric) space.

2.2 Pivoting in the Tableau

By employing Gaussian elimination the problem variables can be expressed in terms of a new set of non-basic (parametric) variables. Completion of this process is referred to as a pivot step. The computations that are required in a pivot step are referred to as a tableau update. Pivot selection consists of finding a parametric variable t_q that can be raised to a positive level to increase the value of the objective function and finding the constraint row that most strictly binds the increase of t_q. From examination of the expression (2.1.3) it is evident that the objective value can increase with an increase in t_q only if $a_{0,q} \leq 0$. This establishes a criteria for selecting a possible pivot column. Any column j for which $a_{0,j} < 0$ may be selected for pivoting. Likewise a problem variable s_k can decrease toward infeasible values with an increase in t_q only if $a_{k,q} > 0$. Thus for any column q the most binding constraint is found by:

$$\min_i (a_{i,0}/a_{i,q}) \quad i=1,\ldots,m+n \quad a_{iq} > 0 \qquad (2.2.1)$$

(a/b denotes a divided by b)

Having selected a pivot column q and a pivot row k, the tableau element $a_{k,q}$ is referred to as the pivot element. The following formulae update the tableau:

column q $\quad a_{i,q} = -(a_{i,q}/a_{k,q}) \quad i=0,1,\ldots,m+n \qquad (2.2.2)$

columns $j \neq q$ $\quad a_{i,j}=a_{i,j}-a_{i,q}(a_{k,j}/a_{k,q}) \quad i=0,1,\ldots,m+n \qquad (2.2.3)$

Geometrically the parametric variables may be viewed as an n-dimensional orthogonal coordinate system. Since the parametric variables are restricted to take on only non-negative values, the optimal solution to the problem always lies in the positive orthant of this coordinate system. The updating operations translate the origin of the parametric coordinate system from one vertex of the convex polyhedron to an adjacent one. The coordinate axes also undergo a rotation such that the convex polyhedron is continually distorted as a result of the pivot operations. The current solution is the vector of values found in the zeroth column when the parametric variables are at zero levels. Hence the current solution always lies at the origin of the parametric coordinate system. At optimality a coordinate system with its origin at the optimal extreme point of the convex polyhedron has been attained.

2.3 The Gomory Cuts

It is evident that the updating formulae (2.2.2, 2.2.3) could not always be expected to result in integer solutions to the problem. However, by examining these formulae one can deduce that such will be the case if all of the entries in the initial tableau are integer and if all pivot elements ($a_{k,q}$ in 2.2.2 and 2.2.3) are selected as ± 1. These requirements are met by the special Gomory cuts that are employed in the G-Y algorithm.

In terms of the tableau representation, Gomory considered a constraint of the type:

$$s_i = a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j) \qquad (2.3.1)$$

and showed that non-negative integer vectors $\underline{t} = (t_1,\ldots,t_n)$ which satisfy (2.3.1) also satisfy:

$$r = = [a_{i,0}/p] + \sum_{j=1}^{n} [a_{i,j}/p](-t_j) \qquad (2.3.2)$$

$$p \neq 0$$

and [a/p] denotes the largest integer less than or equal to a/p.

The G-Y algorithm considers the most binding constraint k for a column q to be (2.3.1). It then selects $p=a_{k,q}$ (the pivot element) so that the cut that is generated will have +1 in the pivot column. The cut (2.3.2) is appended to the tableau in row m+n+1 and the pivot element is changed from $a_{k,q}$ to $a_{m+n+1,q} = +1$. Since the initial tableau has only integer entries, and since the coefficients of (2.3.2) are all integers, integrality is maintained in the updating process. The following is an example of one pivot step on the problem:

$$\begin{aligned} \max \quad & z = 2x_1 + 3x_2 \\ & -x_1 + 2x_2 \leq 5 \\ & x_1 + x_2 \leq 5 \\ & x_1, x_2 \text{ non-negative integers} \end{aligned}$$

The initial tableau is shown in Figure 2.3.1. The pivot element of Figure 2.3.1 was $a_{2,3} = 2°$. The cut r was generated from constraint s_1. The new pivot element $a_{6,3} = 1*$ produces an updated tableau shown in Figure 2.3.2.

		$-t_1$	$-t_2$
z =	0	-2	-3
s_1=	5	-1	2°
s_2=	5	1	1
x_1=	0	-1	0
x_2=	0	0	-1
r =	2	-1	1*

Figure 2.3.1

		$-t_1$	$-t_2$
z =	6	-5	3
s_1=	1	1	-2
s_2=	3	2	-1
x_1=	0	-1	0
x_2=	2	-1	1
r =	0	0	-1

Figure 2.3.2

These operations are shown geometrically in Figures 2.3.3, 2.3.4, and 2.3.5. Figure 2.3.3 shows the original problem constraints. Figure 2.3.4 shows the cut that is added and indicates the move from integer vertex 1 to integer vertex 2. Figure 2.3.5 shows the convex polyhedron in terms of the new parametric coordinate system (notice the distortion of this polyhedron).

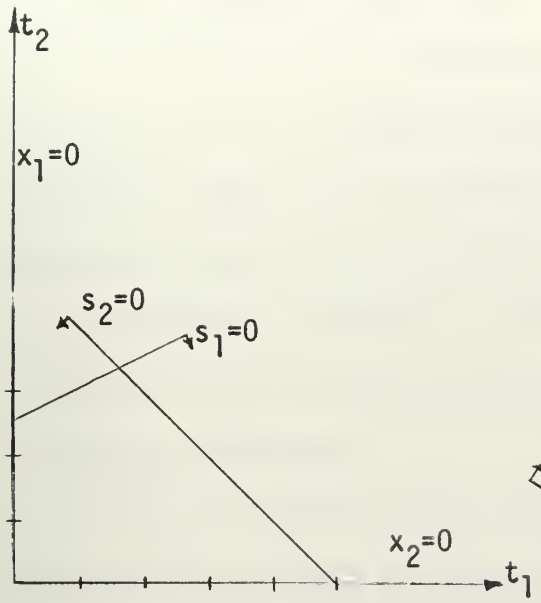

Figure 2.3.3

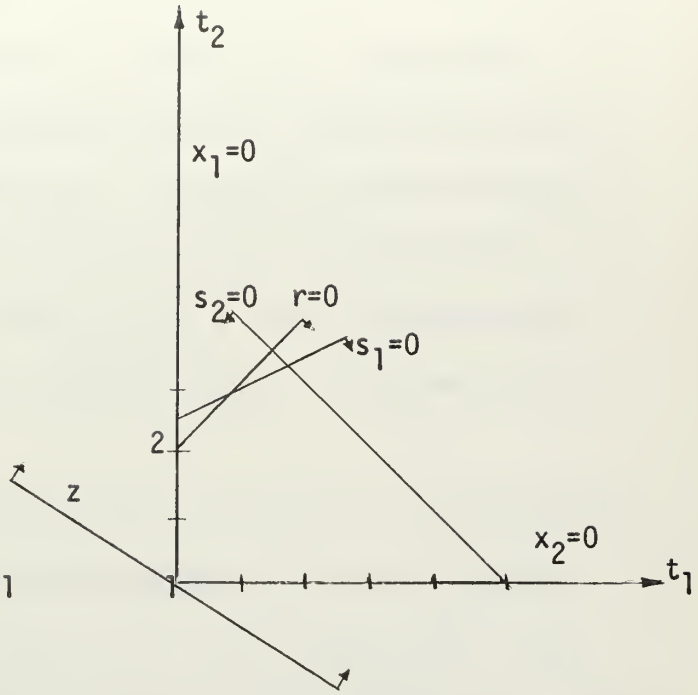

Figure 2.3.4

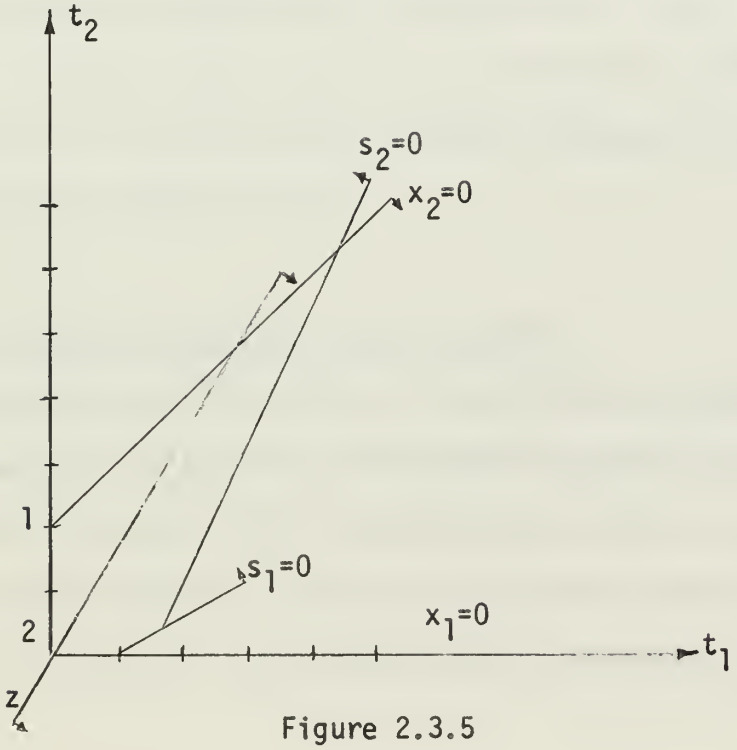

Figure 2.3.5

2.4 Some Properties of the Cuts

If in the process of generating a cut (2.3.2) it occurs that $[a_{k,0}/a_{k,q}] = 0$, the updating operations will not change the previous solution. Such a cut is referred to as a "zero cut" and the updating operations that are involved in pivoting on such a cut are referred to as a "stationary iteration". If $[a_{k,0}/a_{k,q}] \neq 0$ then "breakthrough" is said to have occurred and the resulting update is referred to as a "transition iteration".

The fact that zero cuts can (and often do) arise in the course of the algorithm presents a problem of convergence not unlike that encountered in the normal simplex method for linear programming. (Discussion of this relationship will be postponed until Chapter IV). Young designed a method that takes the possibility of zero cuts into account and guarantees that the algorithm will converge in a finite number of pivot steps. Although Young's proof is too elaborate to reproduce here, it is apparent that zero cuts should be avoided wherever possible since they slow down the solution process.

2.5 Summary of the Basic Gonzales-Young Algorithm

All of the essentials of a basic G-Y algorithm have been presented. Although the convergence proof of the algorithm depends on some rather specialized pivot selection rules, the basic algorithm can be presented in a straightforward manner. To date no problem has been shown to cycle under the basic algorithm if the problem can be guaranteed to be bounded

in every direction (i.e., possesses a closed convex set). The algorithm assumes an all integer tableau.

A. Among the columns j with $a_{0,j} < 0$ select one for pivot consideration. (If there are no $a_{0,j} < 0$, the current tableau is optimal.)

B. Having chosen column q for pivot consideration, select the most binding row k from:

$$\min_i [a_{i,0}/a_{i,q}], \quad a_{i,q} > 0, \quad i=1,\ldots,m+n$$

(If there is no $a_{i,q} > 0$ $i=1,\ldots,m+n$, the problem is unbounded.)

C. From row k generate a cut by setting $p=a_{k,q}$ in (2.3.2).

D. If $[a_{k,0}/a_{k,q}] = 0$, check to see if there are other negative columns that have not been considered. If there are, go back to A. If not, go to E.

E. Append the cut in row m+n+1 of the tableau and select $a_{m+n+1,q} = 1$ as pivot element. Perform the updating operations, erase the cut, and go back to A.

2.6 Phase I of the G-Y Algorithm

It sometimes occurs that the AILP problem cannot be formulated in such a way as to provide a starting basic feasible solution in the initial tableau. There are two forms of infeasibility that can arise in the initial tableau. The first case is when equations instead of inequalities appear in the problem formulation. The second case is when the $\leq$ inequalities have a negative right hand side (i.e. the constraint slack has a negative value in the initial tableau)

Suppose that constraint k is originally formulated as an equation. The slack s_k is still added to this constraint when it is changed to the tableau format. However, any feasible solution to the equation requires that $s_k=0$. Gonzales shows that, by successively pivoting on cuts generated from the row marked s_k , the row can be reduced to the representation $s_k = 0-a_{k,q}(-t_q)$ if there is a feasible integer solution to the equation. At this point it is apparent that t_q can be kept to a zero level simply by forcing it to remain non-basic. Therefore, the column marked $-t_q$ can be dropped from the tableau and subsequent pivot steps assure that s_k remains at a zero level (feasible). These operations are applied to the equations one by one until all have been satisfied.

When any tableau has a negative problem variable (including the slacks), this variable is currently infeasible. Gonzales reasons that the solution space associated with any subset of the problem constraints must include the solution space associated with the complete constraint set. Since the goal is to raise a currently negative slack variable s_k

to a non-negative value, the row marked s_k is temporarily chosen as the objective row to be maximized. This row is maximized using the normal algorithm. However, only the constraints that are currently feasible are considered for pivoting. The set of currently feasible constraints is referred to as the subproblem. The temporary objective function s_k is maximized over this subproblem until s_k reaches a non-negative value. At that point s_k is feasible and its associated row is included in the subproblem. Thus once a variable becomes feasible it is not allowed to go infeasible at a later iteration. Each infeasible variable is treated individually in this manner until all infeasibilities have been removed and the subproblem contains the entire constraint set. At this point attention is returned to the true objective function and the normal algorithm is applied to optimality.

CHAPTER III

COMPUTER IMPLEMENTATION OF THE GONZALES-YOUNG ALGORITHM

One of the primary objectives of the research reported here was to develop a packaged computer program for the solution of AILP problems by the G-Y algorithm. This program has been written in the procedure-oriented ICETRAN (ICES Fortran) language[1]. However, the CDL(Command Definition Language) of ICES[2] has been employed to facilitate user interaction with the program. The whole package has been incorporated as a component (called OPAILP) of the OPTECH subsystem of ICES[3]. OPTECH is the mathematical programming subsystem containing several other techniques such as linear programming and network flow analysis.

OPAILP allows the user to specify his problem in much the same way as he would normally formulate it. A small set of procedure-initiating Commands and their accompanying data are all that are required to define the problem to the computer and to initiate the solution process. In the absence of input error conditions the user can expect to receive his answers in a convenient form. With the exception of the time and number of iterations required for the solution, no data related to the algorithm is output.

This chapter is intended to serve as a user's guide for the OPAILP package. It begins with a brief discussion of the considerations that influenced the design of OPAILP's data structure and

data management philosophy. It then specifies the procedures that are required in the proper use of the program and briefly describes the computer routines that make up the CDB's (Command Data Blocks)[2] for command processing. An example problem that encompasses all of the OPAILP capabilities is presented. Computer source program listings and detailed program documentation are contained in Appendix A of this paper.

3.1 Data Structure Considerations

The dynamic array capability offered by the ICES system allows considerable flexibility in problem data management. The way in which problem data are structured into dynamic arrays can affect the efficiency of data management to a considerable degree. Usually the data structure comes under closest scrutiny when it is possible for data requirements to exceed the storage capacity of primary computer memory. This is so because the inefficiencies encountered in referencing poorly structured core-resident data are small compared to unnecessarily repetitive transfers of data between primary and secondary storage. In general, when one knows in advance the order and frequency of data requirements of the algorithm under consideration, the design of an efficient data structure is a distinct possibility. These requirements are met by the G-Y algorithm.

In its basic form the algorithm has data requirements that are relatively well defined. It consists of the repetitive application of four sequential steps (see Section 2.5). The data that

these steps require are either all members of a particular row or all members of various columns.

A. Pricing- requires data elements from the first row of the tableau. When a negative coefficient is encountered, pricing is halted to see if 'breakthrough' can be achieved.

B. Finding Most Binding Constraint- requires the zeroth column of the tableau and the pivot column that is currently under consideration. Entries in column zero and the pivot column are referenced sequentially from the top of the tableau to the bottom.

C. Pivot Acceptance/Rejection- no additional data is required. If the pivot column currently under consideration generates a zero cut and there are more negative columns to be examined, the current proposal is rejected. When all negative columns have been considered and only zero cuts are available, one of these is accepted for pivoting. Breakthroughs are always accepted.

D. Tableau Updating- requires the referencing of all columns of the tableau. The pivot column is used to update all other columns by the formulae given in Section 2.2.

When data must be referenced through a dynamic array pointer

structure, the number of levels in this structure should be kept to a minimum. Thus there is always a question of how the data elements should be grouped to form larger referencing entities. For example, every step of the G-Y algorithm references data elements as members of a particular column or members of a particular row. The requirement for one member of a column (row) usually implies an impending requirement for all other members of the column (row). Therefore, grouping the data in the form of row entities or column entities seems an obvious possibility. However, there is also the possibility of transmitting even larger blocks of data between primary and secondary storage (e.g. strings of columns or rows). All of these arrangements are possible and the actual steps that the algorithm performs provides the basis for meaningful comparisons among the possible alternatives.

The data structure that has been chosen for OPAILP considers single columns as entities in transferring data between primary and secondary storage. The idea of row entities was rejected out of hand primarily because of Step B presented above. Finding the most binding constraint would require the referencing (and transfer) of every tableau element each time a particular column was considered for pivoting, even though this step requires elements from only two columns for this purpose. Since several zero cuts are often rejected before a pivot step is actually performed, row-wise referencing would require several transfers of every row at each pivot step. It was obvious that row-wise referencing was too inefficient to receive serious

consideration. This left a choice between single columns and strings of columns. This choice was somewhat more difficult, because there was a delicate tradeoff involved. Naturally, the larger the entity, the fewer will be the number of secondary storage references in any given step of the algorithm. However, single column referencing provides some types of flexibility that stringing columns does not allow. To bring this point to the fore, some more detailed considerations of the algorithm must be presented.

There are two aspects of the algorithm that require a considerable degree of flexibility from the data structure. The first of these concerns convergence considerations. Both Young's convergence proof [4] and the convergence heuristics of Gonzales [5] require the logical switching of columns of the tableau. The details of these requirements are too elaborate to present here. However, in general terms, convergence considerations require that, at some particular stage of the computation, column j and column k must logically change places in the tableau so that they can be considered for pivot selection in reverse order. The second aspect of the algorithm that affects the data structure concerns the use of the algorithm as a component in the solution of a MILP problem using a method developed by Benders [6]. In this context the G-Y algorithm would have to add rows to its tableau periodically. The author feels that this will eventually be a very important function of the G-Y algorithm since all of the requirements of Benders' algorithm are currently available in OPTECH except for a required interface

between the LP and AILP routines.

All dynamic array structures are permitted to grow. However, if the columns were to be strung together as transfer entities, the columns within any particular string would not be able to grow conveniently. One would have to either attach unused storage locations at the end of every column in anticipation of the rows that would be added (see Figure 3.1.1) or provide some way to continue each column outside the string (Figure 3.1.2). Either of these approaches would be cumbersome; the first because there is no way to predict the number of rows that would be added, and the second because data management in this structure would be expensive. The logical switching of columns in either of these schemes would have to be done by physically moving them or by an elaborate bookeeping system. On the other hand, single column entities handle both of these problems with relative ease. Figure 3.1.3 shows that unlimited column growth is permitted, and Figure 3.1.4 demonstrates the ease with which two columns can be logically exchanged via the ICETRAN SWITCH command [7]. These considerations led to the choice of single columns as data transfer entities. However, this choice led to a rather detailed data management strategy designed to minimize the cost of storing and retrieving data elements in these relatively small blocks.

3.2 Data Management Strategy

Generally speaking, a good data management strategy will keep

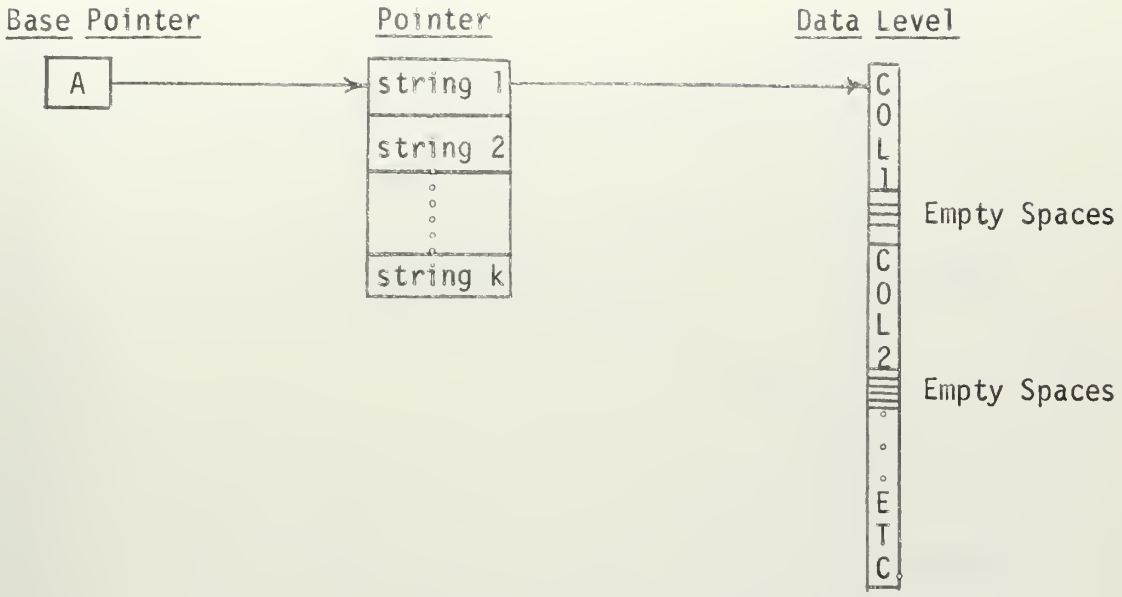

Figure 3.1.1

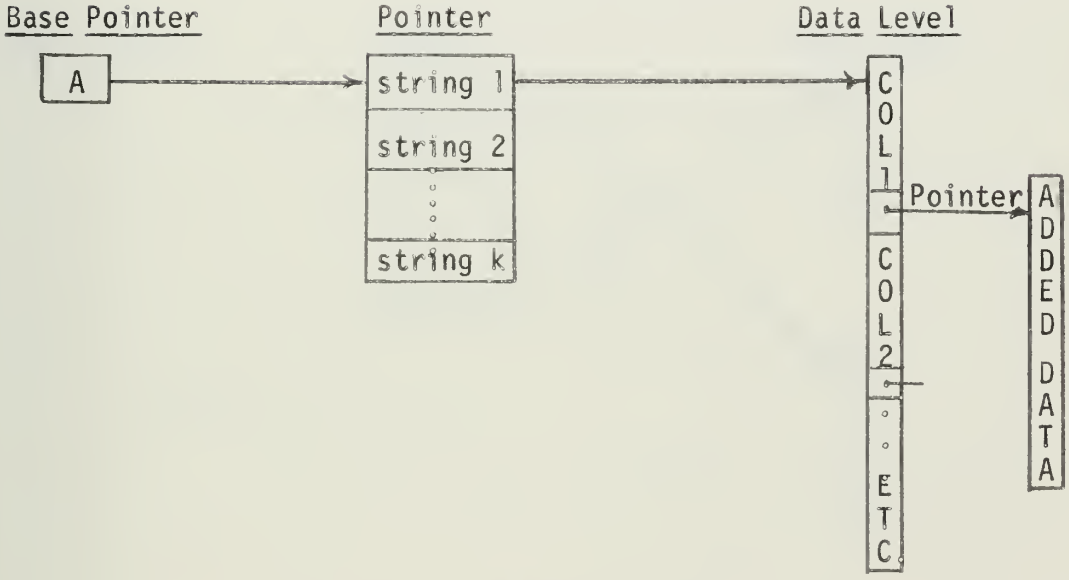

Figure 3.1.2

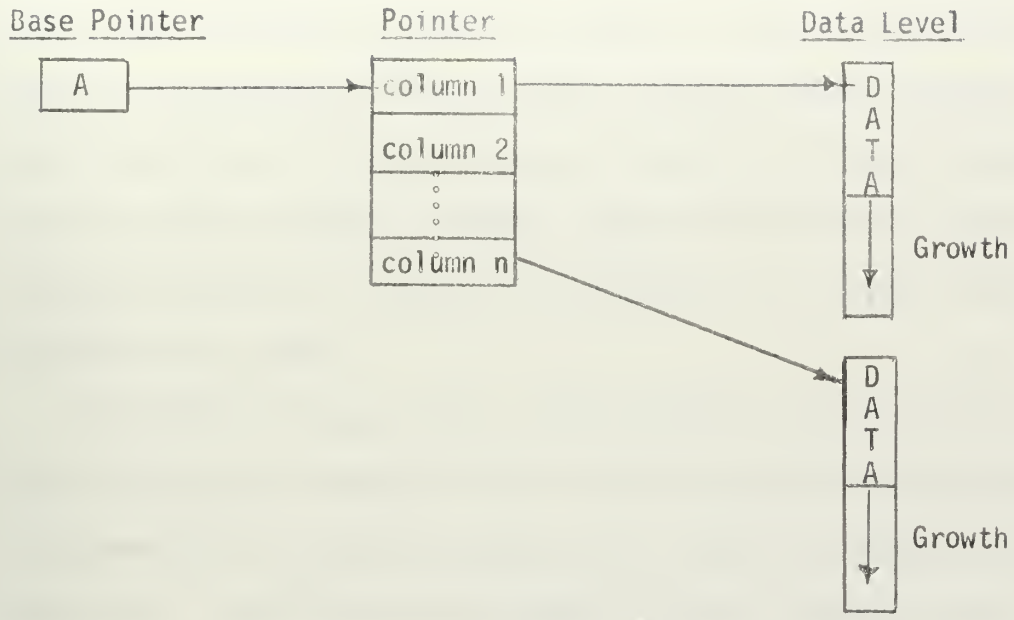

Figure 3.1.3

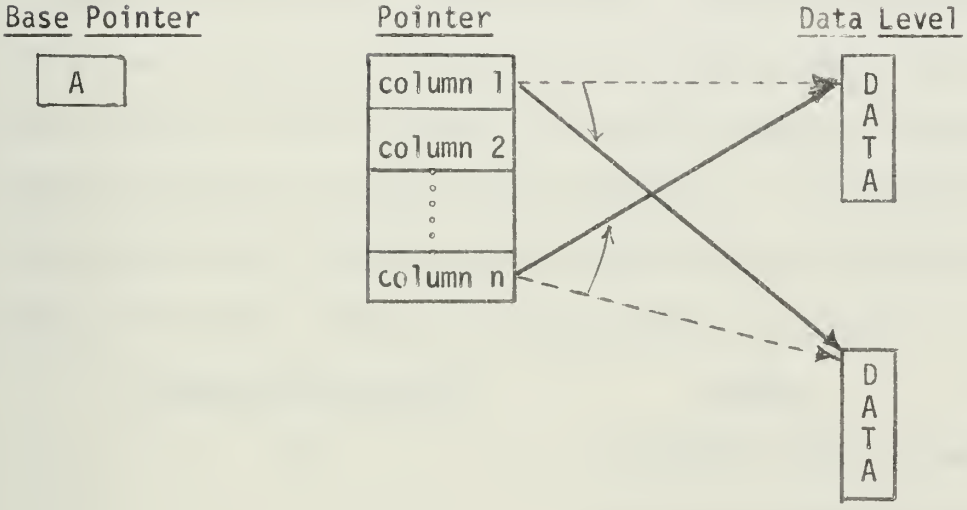

Figure 3.1.4

the most urgently needed data in primary memory so that accesses to secondary storage are kept to a minimum. It will also take into account the potential effects of primary memory reorganizations on the efficient use of the available data pool [8]. It must predict the future need for data of various types and take steps to insure that data handling will result in an optimal use of primary memory reflecting these needs.

The ICETRAN language allows the programmer to assume some of the data management responsibilities usually performed by the system. The programmer is allowed to specify the relative importance of each data entity. Each entity may be given a status (released or unreleased) and a priority (high or low). The data pool may contain all types of data with all combinations of status and priority. However, when new data are transmitted to primary memory with no space available in the data pool, the monitor system is forced to reorganize the pool to create the needed space. Four distinct and increasingly drastic options may be exercised by the system for this purpose. These four 'levels of reorganization' consist of:

Reorganization Level	Action
0	reposition arrays in pool
1	purge low-released arrays
2	purge high-released arrays
3	purge low-unreleased arrays

Reorganization at any level purges all arrays effected by that level.

The data management strategy that is employed by OPAILP utilizes its knowledge of memory reorganization policies in attempting to keep urgently needed data in primary memory. The strategy considers that there are only two basic operations that the G-Y algorithm performs. These are pivot selection (Steps A, B, and C) and updating (Step D). The updating operations require each of the columns (one at a time). This implies that the data can be 'prepared' during the updating process for efficient referencing in the pivot selection operation. The pivot selection operations consider only negative columns. Other columns need never be referenced in this operation. To take advantage of this fact OPAILP keeps a copy of the objective function in a one-dimensional array outside of the regular tableau structure. This array stays at an high-unreleased level so it can never be removed from the data pool. The algorithm searches this one-dimensional array in order to propose pivot columns. Thus a column need never be referenced simply to see if it is negative and as such eligible for pivot selection. Releasing policies within the updating operations insure that there is always a high density of data in the pool. Notice that if all columns were released at the same level in the updating process a memory reorganization would leave the data pool completely empty. This would result in grossly inefficient use of space. To combat this effect the algorithm keeps track of the number of columns that will fit into the pool. It will not release data until that limit is approached. However, when the data pool is judged

to be relatively full, all other columns that are encountered in updating are set to a low-released level. If and when a reorganization is required, it will not occur above Level 1 and space will be freed though the pool will remain relatively full.

3.3 OPTECH/OPAILP Commands and Their Uses

The OPTECH executive routine provides the set of command data blocks used by the ICES executive program in processing OPTECH commands. This executive routine is written in the CDL language of ICES. The OPTECH/OPAILP component encompasses only six of these commands. A user-constructed set of commands, and command-dependent input data, is used to initiate the OPTECH/OPAILP routines. The set of commands listed on the following page, and their associated data, constitute the full capability of OPAILP. The Required commands are the minimal set that can be used to solve an AILP problem. They are presented in the order that they must occur in the user's data deck. Any other ordering of these commands will cause a termination of the job. The Optional commands provide additional capabilities. They also have a required ordering, but this will be discussed in the next section where sequencing and data requirements are specified.

COMMAND	FUNCTION
REQUIRED COMMANDS	
OPTECH	initiates OPTECH subsystem
AILP 'ROW n_1' 'COL n_2' 'LIMIT n_3'	initiates the G-Y procedure The terms in ' ' are optional. They specify the number of rows, the number of columns, and the user's limit on the number of iterations that will be performed.
RIGHT SIDE	initiates the input of rhs data
[rhs data deck]	
TABLEAU	initiates the input of the matrix coefficients
[Matrix data deck]	
ITERATE	initiates solution process Standard output occurs when the process is completed.
FINISH	system command marking the end of the job
OPTIONAL COMMANDS	
INSERT ROW n_1	initiates the addition of a row The row is added to an optimal
[New row data deck]	tableau and a reoptimization is performed.
WRITE (Any OPAILP command)	echo-prints the data accompanying any OPAILP command

3.4 Command Documentation

This section explains the function of each of the OPAILP commands. The underlined portion of each command name constitutes a short form version of the command that is acceptable. All data are read in under fixed format conditions by the CDB's associated with each command. This restriction has been imposed because the volume of data that is usually required would lead to prohibitive input times if the data were to be processed under the free format that is available with the CDL. Particular consideration should be given to these format restrictions since all data is input in either integer or alphanumeric form. In every case the data should be right justified in the data field that is provided.

Command Documentation OPTECH:

1. External command structure:

   OPTECH

2. Description: The system command OPTECH must be the first command in any OPAILP run; it requests the ICES system to transfer control to the OPTECH subsystem (of which OPAILP is a part).

Command Documentation AILP:

1. External command structure:

   AILP ROW n_1 COL n_2 LIMIT n_3

2. Description: The AILP command immediately follows the OPTECH system-level command; it initiates the G-Y AILP procedures.

The optional modifiers are used as follows: ROW n_1 inputs the user's estimate of the number of rows, n_1. COL n_2 inputs the number of columns, n_2. LIMIT n_3 inputs a limit on the maximum number of iterations that the user will permit. If the problem is not solved within this limit, the best available solution will be output.

Though the problem size modifiers are optional, their omission may increase the solution time slightly. The numbers n_1 and n_2 are meant to measure the size of the problem as formulated (and not to estimate the size of the tableau). They are used to initialize the dynamic array structures. In the absence of user specification, standard values of 5 are assumed. Use of the iteration limit (n_3) should be considered. No matter how small the problem, there is no way to estimate the number of iterations that will be required for an optimal solution. A standard value of 100 is assumed when the user does not specifically set n_3.

Command Documentation RIGHT SIDE:

1. External command structure:

   RIGHT SIDE

2. Description: The RIGHT SIDE command initiates the input of right hand side data. These are the b coefficients in the problem formulation of Section 2.1. These data are placed in the zeroth column of the tableau and the row names that accompany each of the coefficients are used to guide the input of matrix data under the TABLEAU command. One rhs coefficient (whether zero or not) must be input for each constraint row. The objective is not input under this command.

3. Format for rhs data deck: Three pieces of data are required for each constraint. These three data are: constraint type (L for $\leq$, G for $\geq$, or a blank field to indicate an equation); row name (a unique name of up to four characters for each row); and the coefficient itself (an integer number). The L or G of inequality type must appear in card column 8. The row name must appear within the field between card columns 9 and 12. The integer coefficient must be right justified to card column 20. The occurrence of any character in card columns 1 through 4 will mark the end of rhs data (e.g., END). Refer to the example problem of Section 3.5.

Command Documentation TABLEAU:

1. External command structure:

   TABLEAU

2. Description: The TABLEAU command initiates the input of matrix coefficients data. These are the $a_{i,j}$ of the problem formulation in Section 2.1. There is a variable name and a row name associated with every matrix coefficient. The row names are matched against row names input with the RIGHT SIDE command in order to position the coefficients in the proper location in the tableau. The variable name defines the names of the problem variables. Tableau input is column dependent and all coefficients that are associated with a particular variable must be contiguous in the tableau data deck (though row names within any column group may occur in any order).

3. Format for Matrix data deck: The variable name can be up to four characters and must appear within the field between card columns 5 and 8. However, the variable name must be positioned in that field in exactly the same manner every time. The row name associated with the tableau coefficient must appear in the field between card columns 9 and 12 and must appear in exactly the same manner as it did in the RIGHT SIDE data deck. The row name associated with the objective function is always OBJ and it must be right justified to column 12. The coefficient itself

must be an integer number right justified to card column 20. Matrix coefficients that have zero values need not, and should not, be included in the TABLEAU data deck. Slack variables are automatically generated for each row. The user should not install slacks before inputting the constraints since this will lead to a Phase I being required (equations). The occurrence of any character in columns 1 through 4 will terminate the input process (e.g., END).

Command Documentation ITERATE :

1. External command structure:

   ITERATE

2. Description: The ITERATE command simply initiates the solution process. Provided the RIGHT SIDE and TABLEAU commands have occurred in proper sequence, the ITERATE command will attempt to maximize the objective function. Output will immediately follow this command whether the optimal solution is achieved or the user's iteration limit is exceeded.

Command Documentation INSERT ROW:

1. External command structure:

   INSERT ROW 'TYPE'

2. Description: The INSERT ROW command may occur at any time after an optimal solution is attained. (Several rows may be added in succession.) Only one row may be added with each INSERT ROW command. This command takes a constraint expressed in terms of the original problem variables and updates it to the optimal tableau representation. If the insertion of a row causes an infeasibility to appear, the tableau will be re-optimized and the new solution will be output. The term 'TYPE' refers to the type of constraint that is being added (e.g., L, G, etc.). The TYPE data is enclosed in single quotes and must be four characters in length (i.e., just as the constraint type was specified in the RIGHT SIDE input).

3. Format for INSERT ROW data deck: The coefficients of a new row may be read in any order. There is a row name and column name associated with every coefficient. The data format for INSERT ROW data is exactly like the data format for the TABLEAU command. However, the rhs value for a row that is being inserted must be given the column name OBJ right justified to card column 8. Every row coefficient (whether zero or non-zero) must be input explicitly and must be right justified to card column 20. Any character appearing in card columns 1 through 4 will terminate the INSERT ROW input data (e.g., END).

Command Documentation WRITE:

1. External structure:

   WRITE (RIGHT SIDE, OR TABLEAU, OR INSERT ROW)

2. Description: The WRITE command may precede any of the commands listed in the parentheses in order to cause the echo-printing of their data decks (e.g., WRITE TABLEAU).

3.5 An Example Problem

In order to provide a better insight into the procedures required in using OPAILP, an illustrative problem will be presented. The function of each command that is used in this problem will be explained. The problem under consideration is the following:

$$\max \quad 4x_1 + 3x_2 + 2x_3$$

$$\text{subj. to} \quad 2x_1 + 3x_2 - x_3 \leq 9$$

$$-3x_1 + 2x_2 + 2x_3 = 12$$

$$4x_1 - 3x_2 + x_3 \leq 20$$

Notice in particular that the second constraint is an equation so that Phase I is required. As a point of interest it should be noted that the pivot steps employed in gaining a feasible solution to this equation result in an infeasible solution to the third constraint requiring the other part of Phase I described in Section 2.6 to be employed. Figure 3.5.1 is a facsimile of the computer printout that will result from proper specification of the problem mentioned above. The function of each of these pieces of data will be explained. The user commands are assumed to have begun in card column 1, though there are no format restrictions.

The first command, OPTECH, causes a transfer of control to the OPTECH subsystem. The AILP command indicates a 3 x 3 constraint matrix and a user imposed limit of 20 iterations. WRITE RIGHT SIDE causes the echo-printing of the rhs data deck. In this data deck notice that the second constraint is specified as an equation by the absence of an L or G in column 8. The constraint row names are 1, 2 and 3. WRITE TABLEAU causes the echo-printing of the matrix data deck. The row name of the objective function is always OBJ while the other row names match the row names of the rhs data deck. The ITERATE command causes the solution procedure to be initiated. The optimal solution follows. Notice that slacks have been insta led and numbered in the order the constraints were input.

The INSERT ROW command initiates the addition of a new $\leq$ inequality:

$$x_1 + x_2 + x_3 \leq 14$$

Note that OBJ is used in the data deck to indicate the column name of the rhs value of this inequality. Since the previous solution does not satisfy this constraint, a reoptimization results and the new optimal solution follows. The fourth problem constraint has been added to the bottom of the tableau and its slack S004 has been generated.

```
OPTECH
AILP ROW 3 COL 3 LIMIT 20
WRITE RIGHT SIDE
RIGHT SIDE
       L   1        9
           2       12
       L   3       20
WRITE TABLEAU
TABLEAU
      X1 OBJ        4
      X1   1        2
      X1   2       -3
      X1   3        4
      X2 OBJ        3
      X2   1        3
      X2   2        2
      X2   3       -3
      X3 OBJ        2
      X3   1       -1
      X3   2        2
      X3   3        1
ITERATE
      FEASIBLE ITERATION    3
      OPTIMALITY HAS BEEN PROVED
      TOTAL ITERATIONS    5
      STATIONARY ITERATIONS    1
       OBJ=   43
      S001=    1
      S002=    0
      S003=    4
        X1=    4
        X2=    3
        X3=    9
INSERT ROW '   L'
     OBJ   4       14
      X1   4        1
      X2   4        1
      X3   4        1
      FEASIBLE ITERATION    1
      OPTIMALITY HAS BEEN PROVED
      TOTAL ITERATIONS    2
      STATIONARY ITERATIONS    0
       OBJ=   29
      S001=    2
      S002=    0
      S003=   15
        X1=    2
        X2=    3
        X3=    6
      S004=    3
```

Figure 3.5.1

3.6 Output Messages

The following is a list of all output messages that OPAILP may print and the situations under which they occur.

1. FEASIBLE AT ITERATION n

   Printed when the algorithm has attained an initial feasible solution

2. OPTIMALITY HAS BEEN PROVED

   Printed immediately when an optimal tableau is achieved

3. TOTAL ITERATIONS n_1

   Printed before output of problem variables

4. STATIONARY ITERATIONS n_2

   Printed before output of problem variables

5. USER ITERATION LIMIT EXCEEDED

   Printed when optimality has not been achieved within the user-supplied iteration limit (or 100 iterations if the user does not specify a limit)

6. COMMAND OUT OF SEQUENCE

   Printed when a command is encountered before it can be processed (Termination always follows.)

7. NO MATCH FOR — name — IN ABOVE CARD

   Printed whenever a column or row name in the TABLEAU or INSERT ROW data decks cannot be matched with a name that has already been input. (This command is always preceded by an echo-print of the card in which the error was found.)

8. TOO FEW (MANY) COEFF. IN INSERT ROW COMMAND

   Printed when INSERT ROW data deck does not supply a row that is compatible with the problem

9. IN ABOVE CARD, ROW (COL) NAME IS REQ.

   Printed when rhs or tableau data card does not supply a row (column) name when needed

10. IN ABOVE CARD — — IS ILLEGAL

    Printed when a rhs data card does not supply an acceptable inequality type(Card precedes the message.)

11. NO INTEGER SOLUTION EXISTS TO EQUATION name

    Printed when Phase I fails to find an integer solution to the equation named on the message

12. THIS PROBLEM IS UNBOUNDED

    Printed when it is determined that there is no upper bound on the value of the objective function (Job is terminated with no output.)

13. THIS PROBLEM HAS NO FEASIBLE SOLUTION

    Printed when Phase I is unable to raise a negative slack to a non-negative level (Job is terminated.)

14. JOB FLUSHED

    Printed with most error conditions to indicate that the entire job has been scrubbed and EXIT has been called.

Any input error that violates a format restriction will result in a FORTRAN error code and control will be taken away from OPAILP. The entire job will be terminated.

3.7 Some Tips on the Use of OPAILP

The G-Y algorithm has particular difficulty with a certain type of problem. The inclusion of a constraint row that contains coefficients

that are rather large and are relatively prime to each other will almost invariably lead to a large number of iterations. The reasons behind this characteristic would require a rather detailed explanation that probably would not interest the user of this subsystem. However, the user should bear this point in mind when formulating his problem. The integer coefficients should be made as small as possible, commensurate with the accuracy of modeling that is required.

The inclusion of equations in the constraint set often produces a problem that is somewhat more difficult to solve than problems with only inequalities. Thus the user should never add slack variables to inequality constraints before they are input to the program. This will initiate a rather expensive (and unnecessary) Phase I attempt to get a starting basic feasible solution.

If the AILP command includes an accurate estimate of the number of rows and columns of the problem, some computation time will be saved. Estimates that are too low will cause the redefinition of dynamic array sizes and this takes additional time.

The WRITE command allows the user to receive output that shows both his problem definition and his results. However, this feature should not be used if its purpose is to check for keypunching errors in the alphanumeric fields. When keypunching mistakes lead to errors

in these fields the program always echo-prints the particular card that contains the error along with a message identifying the mistake. Input-output operations are always expensive and should be avoided where possible.

CHAPTER III FOOTNOTES

[1] ICES: Programmers' Reference Manual, J. C. Jordan (ed.), Department of Civil Engineering, Massachusetts Institute of Technology, October 1967.

[2] IBID.

[3] Ochoa-Rosso, F., et.al., OPTECH Users' Manual, Department of Civil Engineering, Massachusetts Institute of Technology, October, 1967.

[4] Young, R. D., "A Primal (all-integer)Integer Programming Algorithm", Journal of Research of the National Bureau of Standards-B, Mathematics and Mathematical Physics, Vol. 69B, No. 3, July-September, 1965, pp. 213-250.

[5] Gonzales-Zubieta, R. H., On Some Aspects of Integer Linear Programming, Technical Report No. 16, Operations Research Center, Massachusetts Institute of Technology, June, 1965.

[6] Benders, J. F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol. 4, (1962), pp. 238-252.

[7] See [1] above.

[8] Sussman, J. M., Primary Memory Management in ICES: An Engineering Oriented Computer System, School of Engineering, Research Report R67-68, Massachusetts Institute of Technology, November, 1967.

CHAPTER IV

CONVERGENCE DIFFICULTIES IN THE GONZALES-YOUNG ALGORITHM

This chapter presents a geometric interpretation of the G-Y algorithm and isolates various phenomena that cause its computations to be unnecessarily prolonged. These observations are subsequently used in Chapter V to present a revised concept for a primal AILP algorithm and to develop an algorithm which specifically avoids certain inefficiencies occurring in the G-Y algorithm.

4.1 Convex and Restricted Polyhedra

It is a well known fact that associated with any set of linear constraints there is a unique 'convex polyhedron', the vertices of which correspond to the basic feasible solutions to the associated linear program. The 'faces' of this polyhedron are defined by the geometric representation of the problem constraints as hyperplanes in n-dimensional space. The intersection of n-1 hyperplanes in n-dimensional space defines a straight line segment. A line segment that contains two vertices of the convex polyhedron is called an 'edge' of that polyhedron. Dantzig [1] has offered a geometric interpretation of his Simplex method for linear programs in terms of n-dimensional non-basic (parametric) space. This interpretation views a Simplex pivot step as translating the non-basic coordinate system from one vertex of the convex polyhedron to an adjacent

vertex by moving it along the edge of the convex polyhedron connecting the two vertices in question. This translation is accompanied by a rotation of the coordinate axes so that the geometric representation of the convex polyhedron is distorted as a result. This geometric interpretation was displayed in Section 2.3.

When the same set of linear constraints is used in the formulation of an AILP problem the vertices (basic feasible solutions) of the convex polyhedron may no longer be of interest because they may not be integer points. However, all of the integer points that satisfy the constraint set must be contained in the convex polyhedron. Furthermore, associated with any convex polyhedron containing a set of integer points there can be only one 'restricted polyhedron' having the property that every vertex is an integer point from the set and the entire set is contained within. The concept of a restricted polyhedron is a useful concept in a geometric interpretation of the G-Y algorithm. Clearly if all of the constraints corresponding to the hyperplane faces of the restricted polyhedron could be added to the original constraint set a normal application of the Simplex method would result in only integer solutions to the problem. However, the G-Y algorithm recognizes that there are other, less complicated, ways of attaining integer solutions. Rather than attempting to generate actual faces of the restricted polyhedron, the algorithm simply attempts to uncover edges. There

is a whole family of hyperplanes in n-dimensional space ($n > 2$) that contain a given edge of the restricted polyhedron and an infinite number of hyperplanes from this family border closed half-spaces that contain all of the feasible integer points. Pivoting on any n-1 of these hyperplanes in succession will 'uncover' a given edge of the restricted polyhedron. This is not to say that the algorithm has a priori information about the restricted polyhedron. It only means that the scope of the task that the cuts must perform is not as broad as might be anticipated. The G-Y algorithm, like all other cutting plane algorithms, is merely a modified Simplex method. The cuts that are generated function primarily to surpress non-integer solutions that the Simplex method would normally produce. Actually the term 'cut' appears to be something of a misnomer because these additional constraints do not always cut away portions of the convex polyhedron (this topic will be pursued in the succeeding paragraphs). However, it is important to realize that these cuts are the only thing that distinguishes the G-Y algorithm from the Simplex method. Therefore, any difference in efficiency between the G-Y and the Simplex method is purely a function of the effectiveness of the Gomory cuts that are employed in the G-Y algorithm.

4.2 Convergence Difficulties - An Interpretation

As yet no method has been devised by which properties of the restricted polyhedron can be extracted from computations involving

the original constraint set that defines the convex polyhedron. However, it is possible that in some cases the restricted polyhedron of a given constraint set might be very similar to its convex polyhedron. In fact there might be cases in which the two polyhedra coincide. One would surmise that these cases would yield quite readily to the Simplex-like G-Y algorithm. It seems reasonable to assume that the number of iterations required to solve the AILP problem would compare favorably with the number of iterations required by a Simplex solution of the associated LP problem. Investigations based on this premise have shown this assumption to be false. Several small problems were formulated in such a way that the convex and restricted polyhedra were identical. On these problems the G-Y algorithm usually required many more iterations than the Simplex method even though both methods yielded exactly the same integer solutions. These investigations led the author to recognize several important aspect of the G-Y algorithm, each of which contributes to unnecessary inefficiency. The first of these was a recognition that the G-Y algorithm often generates completely superfluous cuts. This topic is taken up in the remainder of the current section. Discussion of the rest of these points will be postponed until Chapter V where they are used to develop a modified algorithm.

One of the most striking things that was observed in the G-Y solution attempts on the problems mentioned above was the

fact that a very high percentage of the cuts that were generated were completely redundant. In other words these cuts excluded no portion of the convex polyhedron. Perhaps more interesting was the fact that these redundant cuts tended to follow one another in succession. In other words the occurrence of one redundant cut was most often followed by a whole string of cuts of the same type. The hyperplanes associated with the redundant cuts in such a string all intersected the (modified) convex polyhedron at the same integer vertex. All of these cuts (except perhaps the first) were zero cuts. Investigations have shown that these redundant cuts can be anticipated and, with major revisions to the algorithm, they can be avoided. Because the geometric interpretation will be so important in understanding the reasoning behind the modifications that will be proposed later, it will be helpful to first examine the effect that this type of cut can have on the rate of convergence of the G-Y algorithm before determining how they can be anticipated.

Recall that the intersection of n hyperplanes in n-dimensional space defines a point in that space. If only n hyperplanes associated with the problem constraints intersect at each vertex of the convex polyhedron no special difficulty arises in solving the LP problem. In this case there are $\binom{n}{n-1} = n$ edges of the convex polyhedron eminating from each vertex. Pivoting in any one of the n columns of the tableau will translate the non-basic (parametric) coordinate system down one of these edges to an adjacent vertex. Pivot selection rules are designed in such a way as to improve the objective function

at every iteration. Since there is only a finite number of vertices there can be only a finite number of iterations before optimality is attained. On the other hand when more than n hyperplanes, say n+q, intersect at a given vertex there are $\binom{n+q}{n-1}$ line segments eminating from that vertex. However, not all of these line segments can be edges of the convex polyhedron. In fact only n+q-1 of these line segments can be edges of that polyhedron. The other line segments lead from the vertex in question out into the infeasible region and an attempt to translate the non-basic (parametric) coordinate system along one of these line segments will be unsuccessful. Proper application of the Simplex pivot rules will prevent such movement and a degenerate solution will result (a change in basis with no change in basic feasible solution). Geometrically speaking Dantzig's lexico-graphic pivot selection rules monitor the attempted movements down the various line segments. They require that the line segments be explored systematically so that no particular movement is ever attempted twice. Since the number of these line segments is finite, eventually one that is an edge of the convex polyhedron must be explored. When this occurs translation of the coordinate system is achieved and again the algorithm is guaranteed to be finite.

Applying the same geometric interpretation to the G-Y algorithm immediately points to the convergence problem. Consider the possibility of redundant cuts of the nature previously described. All of these cuts intersect at the same integer vertex. At most one of them was needed by the algorithm to define that vertex (only the first of them

was generated in pivoting to the vertex in question). Furthermore, none of these redundant cuts can help to define the next edge of the restricted polyhedron so pivoting on these cuts only serves to perpetuate the degeneracy already existing at that integer vertex. The Simplex lexico-graphic pivot selection rules no longer suffice since each new redundant cut adds new line segments at the vertex, none of which are edges of the restricted polyhedron.

Though Young [2] did not address himself to the possibility of redundant cuts in the algoritm he did show that the algorithm could be made to converge in a finite number of iterations. However, it is the rate of convergence of the algorithm and not its ultimate convergence that is of primary interest here. With this in mind it will first be necessary to establish how these redundant cuts arise in the algorithm before methods can be proposed to avoid them. The following section takes up the first of these issues; the second is the topic of Chapter V.

4.3 Occurrence of Redundant Cuts in the Gonzales-Young Algorithm

Recall that each of the problem variables may be viewed as slacks associated with hyperplanes in the n-dimensional non-basic (parametric) space. Each constraint row in the tableau represents such an hyperplane. A constraint (and hence an hyperplane) will be referred to by the name of its slack variable. Points that satisfy a given constraint yield non-negative values in its slack variable. A constraint s_i is redundant to another constraint s_k when all

non-negative points that satisfy s_k also satisfy s_i , but non-negative points that satisfy s_i do not necessarily satisfy s_k. The phrase 'do not necessarily satisfy...' is used because it will allow s_i to be redundant to s_k if it is a scalar multiple of s_k. In avoiding redundant cuts it will also be desirable to avoid cuts that duplicate constraints that are already present in the tableau.

Returning to the notational conventions of Chapter II, the following statement prescribes the circumstances under which redundant cuts are generated by the G-Y algorithm:

"Any cut that is generated in column k from row i is redundant if $a_{i,0}/a_{i,k}$ = integer"

Proof:

Consider the constraint:

$$s_i = a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j) \qquad (4.3.1)$$

and suppose that a Gomory cut is generated from s_i in column k (i.e. $p=a_{i,k}$ in (2.3.2)). That cut would be:

$$r = [a_{i,0}/a_{i,k}] + \sum_{j=1}^{n} [a_{i,j}/a_{i,k}] (-t_j) \qquad (4.3.2)$$

By definition of $[a_{i,j}/a_{i,k}]$, the $a_{i,j}$ $j=0,1,\ldots,n$ in (4.3.1) can be rewritten as:

$$a_{i,j} = a_{i,k} [a_{i,j}/a_{i,k}] + f_{i,j}$$

where: $0 \leq f_{i,j} < a_{i,k}$

Thus (4.3.1) can be rewritten as:

$$s_i = a_{i,k}\left\{[a_{i,0}/a_{i,k}] + \sum_{j=1}^{n} [a_{i,j}/a_{i,k}](-t_j)\right\} + \\ + f_{i,0} + \sum_{j=1}^{n} f_{i,j}(-t_j) \qquad (4.3.3)$$

The following observations can be made about (4.3.3):

(i) the term in $\{\ \}$ is equal to r

(ii) $a_{i,0}/a_{i,k}$ = integer, implies $f_{i,0} = 0$

(iii) $a_{i,k} > 0$ by assumption

therefore (4.3.3) can be written as:

$$s_i = a_{i,k}(r) + \sum_{j=1}^{n} f_{i,j}(-t_j) \qquad (4.3.4)$$

Consider the $t_j \geq 0$ $j=1,\ldots,n$ that satisfy the cut r (i.e. $r \geq 0$). For such points s_i is not necessarily non-negative. In fact if s_i is not an integer multiple of r (i.e. $f_{i,j} = 0$ $\forall j$) then there are points lying on the hyperplane r (r=0) that do not satisfy s_i (i.e. $s_i < 0$).

Now rewrite (4.3.4) in terms of r:

$$r = (1/a_{i,k})\, s_i + \sum_{j=1}^{n} f_{i,j}(t_j) \qquad (4.3.5)$$

And consider $t_j \geq 0$ $j=1,\ldots,n$ that satisfy the constraint s_i.

Such points yield $s_i > 0$ and since all other terms on the right side of (4.3.5) are non-negative, they all satisfy the constraint r as well. Therefore r is redundant to s_i.

Recalling the previous discussion concerning convergence difficulties, the effect of generating and pivoting on the cut r can now be explored. Suppose that the tableau is currently all-integer. Let the superscript i on entries $a_{i,j}$ imply that $a^i_{i,j}$ is an entry in the i^{th} tableau and superscript i+1 imply tableau i+1, etc. By assumption that $a_{i,0}/a_{i,k}$ = integer , it is obvious that $a_{i,0}/a_{i,k} = [a_{i,0}/a_{i,k}]$. Futher observe that pivoting on column k in tableau i will result in the following situation in tableau i+1:

$$s_i = a^{i+1}_{i,0} = a^i_{i,0} - a^i_{i,k}[a^i_{i,0}/a^i_{i,k}] = 0$$

$$r = a^{i+1}_{m+n+1,0} = [a^i_{i,0}/a^i_{i,k}] - [a^i_{i,k}/a^i_{i,k}][a^i_{i,0}/a^i_{i,k}] = 0$$

(refer to updating formulae in Section 2.2)

Thus the cut r is not only redundant to the constraint s_i , but both s_i and r intersect the same vertex of the convex polyhedron (the vertex at the origin of the non-basic coordinate system of tableau i+1). Therefore, pivoting on the cut r in tableau i has added degenerate solutions to the problem in the manner discussed in the previous section.

Having observed the circumstances under which redundant cuts are generated, attention can be turned to the somewhat more interesting

topic of redundant cuts occurring in strings. It seems reasonable to assume that pivoting on one redundant cut somehow leads to the generation of additional redundant cuts in succeeding tableaus. This assumption is correct and the circumstances that lead to this result can be stated as follows:

"If a cut is generated in column k from row i having the property $a_{i,0}/a_{i,k}$ = integer and there are entries $a_{i,j}$ in the row such that $a_{i,j}/a_{i,k} \neq$ integer; each such column j will generate a redundant cut from row i in the succeeding tableau."

Proof:

Consider the tableau updating formulae in Section 2.2. In row i of the updated tableau (i+1):

$$a^{i+1}_{i,0} = a^{i}_{i,0} - a^{i}_{i,k}[a^{i}_{i,0}/a^{i}_{i,k}] = 0 \qquad (4.3.6)$$

and:
$$a^{i+1}_{i,j} = a^{i}_{i,j} - a^{i}_{i,k}[a^{i}_{i,j}/a^{i}_{i,k}] \qquad (4.3.7)$$

But:
$$a^{i}_{i,j}/a^{i}_{i,k} = [a^{i}_{i,j}/a^{i}_{i,k}] + f^{i}_{i,j}$$

So:
$$a^{i+1}_{i,j} = f^{i}_{i,j}$$

And: $a^{i+1}_{i,0}/a^{i+1}_{i,j} = 0$ (integer), and by the previous proof column j generates a redundant cut from row i in tableau i+1.

Thus it has been seen that redundant cuts not only create degenerate solutions, but also can lead directly to the generation of still more redundant cuts and degeneracy is perpetuated through the algorithm. With this thought in mind one can easily appreciate the magnitude of the effort represented in Young's proof of finite convergence. However, it is still reasonable to ask why the generation of redundant cuts is tolerated in the algorithm and to ask whether there might be some way to avoid them.

Suppose that at some stage of the algorithm a constraint s_i is encountered such that s_i would generate a redundant cut r. It has already been seen that both r and s_i intersect a common vertex and r is redundant to s_i. It is logical to ask why the algorithm could not simply neglect to generate r and pivot on s_i instead. The reason that this alternative is not taken is that $a_{i,k}$ (the pivot element) is not necessarily equal to +1 and though the updated tableau would have an integer solution, the other tableau entries would not be integers. Therefore, succeeding pivot steps could not be guaranteed to produce integer solutions. At that point the algorithm (in its present form) would break down.

The possibility of avoiding redundant cuts in the manner described above led the author to search for methods for restoring the integrality of the tableau once it has been lost. Such a method was discovered, but it was found to generate redundant cuts much in the same way that was described above, so it was abandoned. However, the concept of this method was derived directly from the author's realization that

the G-Y algorithm is rather strongly related to Gomory's Algorithm I (a dual feasible method) though on the surface the two algorithms seem completely dissimilar. This observation has led the author to propose a modified primal AILP algorithm that draws from both of the algorithms mentioned above. This observed relationship between the two algorithms will be presented first in Chapter V before the modified algorithm is described.

CHAPTER IV FOOTNOTES

[1] Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, N. J., 1963.

[2] Young, R. D., "A Primal (all-integer) Integer Programming Algorithm", Journal of Research of the National Bureau of Standards-B, Mathematics and Mathematical Physics, Vol. 69B, No. 3, July-September, 1965, pp. 213-250.

CHAPTER V

A MODIFIED PRIMAL AILP ALGORITHM

This chapter draws on the previous geometric interpretation of the G-Y algorithm in order to motivate the reasoning behind the modifications that are proposed. It was seen in Chapter IV that the G-Y algorithm tolerates the generation of redundant cuts in order to preserve an all-integer tableau. The present chapter shows that an all-integer tableau is not essential to the concept of a primal AILP algorithm. Methods are presented here that not only handle the non-integer tableau, but also specifically avoid the generation of redundant cuts and their accompanying inefficiencies. Gomory cuts remain the vehicle by which integer solutions are produced, so essentially no new concepts are introduced in the context of Integer Linear Programming. However, since the cuts that are used in the modified algorithm are developed in a manner somewhat different from the way cuts are developed in the G-Y algorithm, it will be necessary first to present these cuts in their original context (Gomory's Algorithm I). Then, in order to bring the geometric interpretation abreast of the theoretical development, this chapter shows how the different cuts can be used to produce a primal algorithm that is equivalent to the G-Y algorithm. This chapter goes on to generalize the concept of this equivalent algorithm to produce a modified primal AILP algorithm. This modified algorithm is compared to various other

AILP algorithms (including G-Y) on several problems. Finally an attempt is made to explain the different rates of convergence among the algorithms and to place the modified algorithm in perspective with respect to other cutting plane methods.

5.1 Review of Gomory's Algorithm I

Chapter I traced the development of cutting plane methods to the present. Recall that Gomory's Algorithm I was the first proven algorithm for the AILP problem. This algorithm is a dual feasible method in that no integer solution is produced until the optimum is found. The integer programming portion of the algorithm begins with an optimal tableau for the associated LP problem. The generation of each cut is followed by a complete reoptimization of the newly defined (by virtue of the new constraint) LP problem. Optimality is achieved when the reoptimized LP problem has an all-integer solution.

From the optimal tableau of the LP problem Gomory selected a row that represented one of the integer constrained problem variables whose value was non-integer.

$$x_k = a_{k,0} + \sum_{j=1}^{n} a_{k,j}(-t_j) \qquad (5.1.1)$$

and showed that non-negative integers t_j $j=1,\ldots,n$ that satisfy (5.1.1) also satisfy:

$$r = -d_{k,0} + \sum_{j=1}^{n} -d_{k,j}(-t_j) \qquad (5.1.2)$$

where: $d_{k,j} = a_{k,j} - [a_{k,j}]$ $\quad 0 \leq d_{k,j} < 1$

and $[a_{k,j}]$ denotes the largest integer $\leq a_{k,j}$
($a_{k,j}$ typically non-integer)

He also showed that r was constrained to take on non-negative values by virtue of the way it was derived. When (5.1.2) was appended to the tableau it made the previous optimal solution to the LP infeasible and a reoptimization was required. Geometrically speaking the cut (5.1.2) actually cut the previous optimal solution away from the convex polyhedron in the same way that the cut (2.3.2) is expected to function in the G-Y algorithm. However, it is evident that (5.1.2) could never be a redundant cut since $d_{k,0}$ could never be zero. The vertex of the convex polyhedron that corresponds to the basic feasible solution of the tableau from which r is generated is always excluded by the inclusion of r in the constraint set.

5.2 On the Relationship Between the G-Y Algorithm and Algorithm I

Since the two types of cuts that have been discussed serve roughly the same purpose in the algorithms that generate them it seems reasonable to assume that they are more strongly related than their derivations tend to indicate. The discussion that follows will show that this assumption is correct.

Recall that discussions in Chapter IV reached the conclusion that the G-Y algorithm must tolerate redundant cuts in order to

preserve integrality of the tableau. It was seen there that these redundant cuts could be anticipated and could be avoided if methods were available to restore the tableau's integrality. In searching for such a method the author discovered that cuts used in Gomory's Algorithm I (5.1.2) could be helpful in this effort. It was also observed that these cuts could be used in a primal context to produce an algorithm that is equivalent to the G-Y algorithm. This equivalent algorithm starts with an all-integer tableau just as the G-Y algorithm. However, it requires two distinct pivot steps to achieve the same results that are achieved in the G-Y algorithm in one pivot step. The equivalent algorithm first performs a normal Simplex pivot step in the first tableau. Then from the updated tableau a cut (5.1.2) is generated in a special way. This cut turns out to be exactly the same cut as the one that would have been generated by the G-Y algorithm in the initial tableau. Therefore, pivoting on this cut in the updated tableau produces exactly the same results as one pivot step of the G-Y algorithm. This procedure will be presented in detail below, but first one important point should be considered. Although the equivalent algorithm requires two pivot steps for every one required by the G-Y algorithm, its procedure does demonstrate an ability to handle a non-integer tableau. Thus the equivalent algorithm is presented here only to motivate the concepts that will be generalized later to produce a modified primal AILP algorithm.

Suppose that an initial all integer tableau is given and that the G-Y algorithm has decided to generate a cut (2.3.2) in column k from the constraint:

$$s_i = a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j) \tag{5.2.1}$$

that cut would be

$$r = [a_{i,0}/a_{i,j}] + \sum_{j=1}^{n} [a_{i,j}/a_{i,k}]\ (-t_j) \tag{5.2.2}$$

However, suppose that instead of generating and pivoting on (5.2.2) a normal Simplex pivot step was performed by selecting $a_{i,k}$ as the pivot element. This would raise the parametric variable t_k to a positive level (bring it into the basis) and replace it in the non-basic (parametric) variable set with t_k^*. Observe the effect of this pivot step on the tableau representation of the variables t_k and s_i. In the initial tableau:

$$t_k = 0 + (-1)(-t_k) \tag{5.2.3}$$

$$s_i = a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j) \tag{5.2.4}$$

In the updated tableau:

$$t_k = a_{i,0}/a_{i,k} + \sum_{\substack{j=1 \\ j\neq k}}^{n} a_{i,j}/a_{i,k}(-t_j) + (1/a_{i,k}(-t_k^*) \qquad (5.2.5)$$

$$s_i = 0 + (-1)\ (-t_k^*) \qquad (5.2.6)$$

Now if a cut (5.1.2) is generated from the updated tableau representation of t_k that cut would be:

$$r' = -d_{i,0} + \sum_{\substack{j=1 \\ j\neq k}}^{n} (-d_{i,j})(-t_j) + (-1/a_{i,k})(-t_j^*) \qquad (5.2.7)$$

where $d_{i,j} = a_{i,j}/a_{i,k} - [a_{i,j}/a_{i,k}]$ (5.2.8)

We wish to show that the cut (5.2.7) is exactly the same as the cut (5.2.2) (i.e., r'=r). Replace the $d_{i,j}$ in (5.2.7) with the right side of (5.2.8). From (5.2.6) observe that $t_k^* = s_i$ and substitute the right side of (5.2.4) for t_k^* into (5.2.7) to yield:

$$r' = [a_{i,0}/a_{i,k}] + \sum_{\substack{j=1 \\ j\neq k}}^{n} [a_{i,j}/a_{i,k}](-t_j) \ - a_{i,0}/a_{i,k} - \cdots$$

$$\cdots - \sum_{\substack{j=1 \\ j\neq k}}^{n} a_{i,j}/a_{i,k}(-t_j) + (1/a_{i,k})\left\{a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j)\right\} \qquad (5.2.9)$$

Cancellation of terms in (5.2.9) yields:

$$r'=[a_{i,0}/a_{i,k}]+\sum_{\substack{j=1\\ j\neq k}}^{n}[a_{i,j}/a_{i,k}](-t_j)\ +(a_{i,k}/a_{i,k})(-t_k) \qquad (5.2.10)$$

and since $(a_{i,k}/a_{i,k}) = [a_{i,k}/a_{i,k}] = 1$, (5.2.10) simplifies to $r' = r$. Furthermore, pivoting in column k on the cut (5.2.7) will produce the following tableau representations:

$$t_k=a_{i,0}/a_{i,k}-d_{i,0}+\sum_{\substack{j=1\\ j\neq k}}^{n}(a_{i,j}/a_{i,k}-d_{i,j})(-t_j)+\ldots$$
$$\ldots+(a_{i,k}/a_{i,k})(-t_k) \qquad (5.2.11)$$

$$s_i = a_{i,k}(d_{i,0}) + a_{i,k}\sum_{\substack{j=1\\ j\neq k}}^{n} d_{i,j}(-t_j) - a_{i,k}(-t_k) \qquad (5.2.12)$$

or by substituting the right side of (5.2.8) for $d_{i,j}$:

$$t_k = [a_{i,0}/a_{i,k}] + \sum_{\substack{j=1\\ j\neq k}}^{n} [a_{i,j}/a_{i,k}](-t_j)\ + (-t_k) \qquad (5.2.13)$$

$$s_i = a_{i,0} - a_{i,k}[a_{i,0}/a_{i,k}] +\ldots$$
$$\ldots+ \sum_{\substack{j=1\\ j\neq k}}^{n} (a_{i,j}-a_{i,k}[a_{i,j}/a_{i,k}])(-t_j)\ + (-t_k) \qquad (5.2.14)$$

But recall from Chapter II that:

$$a_{i,j} = a_{i,k}\,[a_{i,j}/a_{i,k}] + f_{i,j}$$

So (5.2.14) can be rewritten as:

$$s_i = f_{i,0} + \sum_{\substack{j=1 \\ j \neq k}}^{n} f_{i,j}(-t_j) + (-t_k) \qquad (5.2.15)$$

By observing the pivot formulae (2.2.2, 2.2.3) one can see that (5.2.13) and (5.2.15) are exactly the same results that would have been attained by pivoting on the cut proposed by the G-Y algorithm in the initial tableau. Thus it is seen that, by using the cuts that are generated in Gomory's Algorithm I in a special way, a primal algorithm can be produced that is equivalent to the G-Y algorithm. This equivalent algorithm works with a non-integer tableau in the second step of its procedure. As an aid to understanding the somewhat cumbersome development presented above, Figures 5.2.1 and 5.2.2 illustrate the parallel between the two techniques. The two tableaus of Figure 5.2.1 reproduce the G-Y pivot step presented in Chapter II. The three tableaus of Figure 5.2.2 show the procedure of the equivalent algorithm.

		$-t_1$	$-t_2$
$z =$		-2	-3
$s_1 =$	5	-1	2°
$s_2 =$	5	1	1
$x_1 =$	0	-1	0
$x_2 =$	0	0	-1
$r =$	2	-1	1*

		$-t_1$	$-t_2$
$z =$	6	-5	3
$s_1 =$	1	1	-2
$s_2 =$	3	2	-1
$x_1 =$	0	-1	0
$x_2 =$	2	-1	1
$r =$	0	0	-1

Figure 5.2.1

		$-t_1$	$-t_2$
$z =$	0	-2	-3
$s_1 =$	5	-1	2*
$s_2 =$	5	1	1
$x_1 =$	0	-1	0
$x_2 =$	0	0	-1

	$-t_1$	$-t^*_2$	
$z =$	15/2	-7/2	3/2
$s_1 =$	0	0	-1
$s_2 =$	5/2	3/2	-1/2
$x_1 =$	0	-1	0
$x_2 =$	5/2	-1/2	1/2°
$r' =$	-1/2	-1/2	-1/2*

		$-t_1$	$-t_2$
$z =$	6	-5	3
$s_1 =$	1	1	-2
$s_2 =$	3	2	-1
$x_1 =$	0	-1	0
$x_2 =$	2	-1	1
$r' =$	0	0	-1

Figure 5.2.2

5.3 Revised Concept of a Primal AILP Algorithm

Although the equivalent algorithm developed in the previous section is interesting from a theoretic standpoint, it is useless as a computational technique. It produces exactly the same results as the G-Y algorithm while requiring twice as much computation. Furthermore, the equivalent algorithm would suffer from the same inefficiencies, such as redundant cuts, that are inherent in the G-Y algorithm. The one redeeming factor in this equivalent algorithm is that it embodies the concepts essential to the handling of a non-integer tableau in a primal approach. Discussion of how these concepts may be employed to overcome some of the inefficiencies of the G-Y algorithm is reserved for the next section of this chapter. However, before pursuing this issue further, the present concept of a primal AILP algorithm must be revised.

Recall from Chapter I that the primary need for a primal AILP algorithm was due to the generally unreliable character of the algorithms that were available. Computational experience with these algorithms showed that it was impossible to predict an upper limit for the number of iterations required for the solution of any particular problem. This characteristic applies equally to the G-Y algorithm. However, the main virtue of this algorithm is that it produces intermediate feasible integer solutions during the computation process. When computation costs prohibit a continuation to the optimum solution, the best of the intermediate solutions might be

useful. Indeed, a relatively 'good' solution is better than no solution. On the other hand it is not strictly necessary for all intermediate solutions to be integer. The main concern is that feasible integer solutions should recur in the solution process and that the frequency of their recurrence can be predicted. It is in this context that the modified algorithm presented below will be referred to as a 'primal' method. All solutions are primal feasible, but not all of them are integer.

5.4 Elements of a Modified Primal AILP Algorithm

The AILP algorithm that is developed here is very similar to the equivalent algorithm developed in Section 5.2. Thus the rudimentary concepts of the different types of cuts will not be repeated. However, several additional details must be supplied before the modifications can be proposed. The first of these details establishes the capability of restoring a non-integer tableau to all-integer entries at any stage of the Simplex method:

"If the initial tableau is all-integer, a Simplex solution procedure can be stopped at any stage of the computations and a series of Gomory cuts (5.1.2) can be employed to restore integrality."

Proof:

Suppose that at some stage of the Simplex computations a matrix B has been inverted. The inverse of B can be expressed as:

$$B^{-1} = \frac{B^{+}}{\det B}$$

where:

B^{+} = the adjoint of B (which must be integer)

$\det B$ = determinant of B (product of pivot elements 1 through i-1)

Let $D^{i} = \det B$ (in the i^{th} tableau)

$|D^{i}|$ = absolute value of D^{i}

Every entry in tableau i can be expressed as:

$$a^{i}_{i,j} = H/D \qquad H,D \text{ integer}$$

A cut (5.1.2) generated from tableau i would have a pivot element of the form:

$$-d_{i,j} = -(h/|D^{i}|) \qquad h,D \text{ integer}$$

$$0 \leq h < |D^{i}|$$

As a result of pivoting on $-d_{i,k}$ in tableau i, tableau i + 1 would have

$$D^{i+1} = (h/D^{i})\, D^{i} = \pm h$$

Obviously since h is strictly less than D^i by an integer amount, at most $n \leq |D^i| - 1$ pivot steps could be required to begin with tableau i and produce an all-integer tableau (because $D^{i+n} = \pm 1$).

The above discussion seems to imply that one could start a Simplex solution of the LP problem and, by keeping track of D, decide to interdict with the cuts to produce an integer solution at an "opportune" moment. This is true, but unfortunately the integer solution could not be guaranteed to be feasible. Such an approach, though appealing, is perhaps too ambitious. However, further restrictions can be applied to preserve feasibility.

Suppose that the Simplex method is begun with an all-integer tableau. The Simplex procedure is allowed to continue until a non-integer solution results (integrality in the rest of the tableau is of no concern). Geometrically speaking this transition from integer vertex to non-integer vertex is pictured as a move along an edge of the convex polyhedron. Recall that movement along this edge has resulted from pivoting in a certain tableau column k. Let $\underline{a}_0^i$ be the integer column vector of values of the problem variables in tableau i and $\underline{a}_0^{i+1}$ the non-integer vector that results from pivoting in column k of tableau i. ($a_{0,0}^i$ and $a_{0,0}^{i+1}$ are the associated values of the objective function).

"If a Simplex pivot in column k causes a transition from a previously acquired integer solution in tableau i to a non-integer solution in tableau i+1, at most $n = |D^{i+1}| - 1$ pivots in column k

on cuts (5.1.2) are required to regain a feasible integer solution having the property $a^{i+n+1}_{0,0} \geq a^{i}_{0,0}$."

Proof:

Consider the Simplex pivot from tableau i to tableau i+1

$$\underline{a}_0^{i+1} = \underline{a}_0^{i} - \lambda\, \underline{a}_k^{i}$$

where λ is a positive scalar and $\underline{a}_k^{i}$ is a lexico-negative pivot column by virtue of the generalized Simplex pivot rules. In tableau i+1 $\underline{a}_k^{i+1}$ becomes lexico-positive. Successive pivots in column k on cuts (5.1.2) will change the solution vector $\underline{a}_0$ in the following way:

$$\underline{a}_0^{i+j} = \underline{a}_0^{i+j-1} - \alpha_{i+j-1}\,(\underline{a}_k^{i+j-1}) \qquad (5.4.1)$$

where α is a positive scalar and $\underline{a}_k$ is always a lexico-positive column vector. Thus (5.4.1) shows that the successive $\underline{a}_0$ form a series of lexico-decreasing solution vectors beginning with $\underline{a}_0^{i+1}$. However, the values that $\underline{a}_0^{i+m+1}$ may take are bounded from below (in a lexico-graphic sense) by $\underline{a}_0^{i}$ because $\alpha_{i+1} + \alpha_{i+2} + \ldots + \alpha_{i+m} \leq \lambda$ (the Gomory cuts are specifically guaranteed not to exclude the integer solution $\underline{a}_0^{i}$). This establishes that $a^{i+m+1}_{0,0} \geq a^{i}_{0,0}$ and $\underline{a}_0^{i+m+1}$ = integer $m+1 \leq n$ and proves the proposition.

Notice that the proof just given immediately allows one to avoid the redundant cuts encountered in the G-Y algorithm. When a cut in the G-Y algorithm would be generated from a constraint s_i such that

$a_{i,0}/a_{i,k}$ = integer, the cut could be neglected and the constraint s_i could be chosen as the pivot row. The next solution would be integer and the rest of the tableau non-integer. When later G-Y pivot steps produced a non-integer solution one could revert to cuts (5.1.2) to regain an integer solution (they can never be redundant). This is one concept of a modified algorithm. Such an algorithm would employ both of the types of cuts that have been discussed. In the general case the cuts (5.1.2) would no longer be equivalent to the cuts (5.2.2) as before. This is so because one would no longer generate them in the very specialized manner prescribed by the equivalent algorithm. However, there is another factor present that has led the author to conclude that the cuts (5.2.2) should be abandoned completely in favor of a generalization of the equivalent algorithm. This conclusion is based on the fact that there is an alternative to the cut (5.1.2) that is more effective.

5.5 Alternative Cuts in the Modified Algorithm

Recall the geometric interpretation of the function of the cut (5.1.2). In any given tableau the current solution lies at the origin of the non-basic (parametric) coordinate system. The cut (5.1.2), when appended to that tableau, makes the origin of non-basic space infeasible (i.e., it cuts away the origin). The effectiveness of any cut depends largely on how much of the convex polyhedron it excludes. Therefore, one would like to generate a cut whose intersections with

the various coordinate axes lie as far away from the origin as possible. One cut is uniformly 'deeper' than another cut when its intersections with the coordinate axes are uniformly farther from the origin.

Hadley [1] has presented a cut, derived from (5.1.2), that is at least as deep and sometimes can be uniformly deeper than (5.1.2). He considered the cut:

$$r' = -d_{i,0} + \sum_{j=1}^{n} (-d_{i,j})(-t_j) \qquad (5.5.1)$$

and showed that if there are any $d_{i,j}$ having the property $d_{i,j} > d_{i,0}$, the term $(d_{i,0})(1-d_{i,j})/(1-d_{i,0})$ can be substituted for $d_{i,j}$ in (5.5.1) to yield an alternative cut that is deeper in the t_j direction for every column j in which the conditions are met. In the limited number of problems on which the modified algorithm has been tested, these deeper cuts have always accelerated the convergence process. However, these deeper cuts should be used with care. Modification of the cut (5.5.1) yields a constraint for which the slack r' is no longer restricted to integer values. To date the author has used a heuristic device to overcome this difficulty. Recall that the term $d^i_{i,0}$ can be expressed as h^i_0 / D^i, $0 \leq h^i_0 < |D^i|$. Once the modification has been applied to (5.5.1) the resulting cut has been multiplied by the scalar $(|D^i| - h^i_0)$ to yield deeper cuts that have 'worked' on the problems tested. This is not to say that such a procedure will

restrict r' to integer values. Indeed, no theoretical basis has been established for this technique; it was developed strictly through trial and error. Actually the whole topic may be of little consequence because the inclusion of the modified cut simply converts an AILP problem into a MILP problem. Chapter VI presents an algorithm very similar to the modified algorithm that will handle the MILP problem. The decision on how to handle these deeper cuts can only come with more experience than is presently available.

5.6 Pivot Selection Rules for the Modified Algorithm

There is one final point that should precede a formal statement of the modified algorithm. This point relates to a modification of the Simplex pivot rules that should be used to accompany this algorithm. Close scrutiny of the proofs in Section 5.4 will show that it is possible to make a complete cycle from an integer solution to a non-integer solution and back to the same integer solution. This process is called a minor cycle. A minor cycle of the modified algorithm is roughly equivalent to a zero cut in the G-Y algorithm. The possibility that this phenomenon can occur in the modified algorithm presents a problem for convergence that has not been overcome to date. However, computational results on the limited number of problems that have been attempted have been very favorable in comparison to other AILP algorithms and no convergence difficulties have been experienced.

Surely one would want to avoid minor cycles whenever possible. It will be shown below that the impending occurrence of a major cycle can be recognized in the algorithm. This fact will motivate a change in the Simplex pivot selection rules so that major cycles are produced whenever possible.

"If in tableau i a Simplex pivot is selected on row i in column k such that $a_{i,0}/a_{i,k} \geq |D^i|$, a major cycle is eminent."

Proof:

Recall that every entry in tableau i can be expressed as H/D^i, H,D integer. Let $a_{i,0}/a_{i,k} = \lambda$, $\lambda \geq |D^i|$.

Thus:

$$\underline{a}_0^{i+1} = \underline{a}_0^i - \lambda\, \underline{a}_k^0$$

In tableau i+m, $\underline{a}_k^{i+m}$ $m = 1,2,\ldots, \leq n$ are lexico-positive.

and successive cuts (5.1.2) yield a series of lexico-decreasing solution vectors $\underline{a}_0^{i+m}$ by virtue of the expressions developed in Section 5.4.

And:

$$\underline{a}_0^{i+m} = \underline{a}_0^{i+1} - (\alpha_{i+1} + \alpha_{i+2} + \ldots \alpha_{i+m-1})\, a_0^{i+1}$$

However, notice that that is another feasible integer solution:

$$\underline{a}_0 = \underline{a}_0^i - \delta|D^i|\ \underline{a}_k^i \qquad (5.6.1)$$

$$\delta = [\lambda/|D^i|]$$

that is lexico-larger than $\underline{a}_0^i$. From this we can deduce that

$$(\alpha_{i+1} + \alpha_{i+2} + \cdots + \alpha_{i+m}) \leq \lambda - \delta|D^i|$$

since the Gomory cuts are guaranteed not to exclude the integer solution (5.6.1). Integrality must be restored in $n \leq |D^i|$ pivot steps and (5.6.1) must be the resulting integer solution.

In order to give preference to Simplex pivot steps that are known to initiate major cycles one should select a lexico-negative column k for which $a_{i,0}/a_{i,k} \geq |D^i|$ if one is available. This is equivalent to the technique used in the G-Y algorithm to avoid zero cuts.

5.7 Computational Results

Obviously the derivations of the previous sections show that there are several alternatives available in both cut generation and pivot selection. The following is a formal statement of the combination of rules that have proved to be the most effective. (An all integer initial tableau is assumed.):

A. In tableau i select the most negative column k that has a most binding row i with $a_{i,0}/a_{i,k} \geq |D^i|$, $a_{i,k} > 0$ (If there are no negative columns, tableau i is optimal.) (If there is no $a_{i,k} > 0$ the problem is unbounded.)

B. Perform a Simplex pivot step on the element $a_{i,k}$.

C. If tableau i+1 has a non-integer solution, select a particular row that displays a non-integer value and generate the cut (5.1.2). GO TO D. If tableau i+1 has an integer solution, GO TO A.

D. If the conditions described in Section 5.5 are met, derive a deeper cut. GO TO E.

E. Pivot on the cut in column k. Erase the cut. GO TO C.

Table 5.6.1 is a summary of results of the modified algorithm when compared to other AILP algorithms. This table was produced in part from data reported by Gonzales [2]. The test problems are ones that are designed to be relatively hard [3].

An interesting fact that was noted in the test problems was that the cases where the modified algorithm shows a markedly better performance over that of the G-Y algorithm was not limited to the problems on which G-Y generates redundant cuts. This is primarily due to the fact that the deeper cuts discussed above sometimes prove to be much more efficient than the specialized cuts of the G-Y algorithm. However, it should be pointed out that the problems on which the algorithms were compared were designed (by Thompson [3]) to be difficult for the algorithms that were in existence at that time. Thus the tests cannot be considered to provide conclusive evidence in favor of any particular algorithm of the group. In view of the

TABLE 5.6.1

DIMENSIONS	ITERATIONS REQUIRED			
OF TABLEAU	ALGORITHM II	THOMPSON	G-Y	HOPPER
5 x 3	13	20	8	6
7 x 4	1000*		37	25
7 x 4	1570	173	84	
5 x 3		255	102	3
5 x 3		260	202	
5 x 3		255	202	

* Indicates that the program was stopped after this number of iterations without having found the answer.

All of the algorithms require roughly the same amount of time to perform an iteration. They all consist of Simplex pivot steps.

fact that certain of the available algorithms give their best performances on specific classes of problems [4], many more tests on practical problems would have to be performed before judgements can be made on the suitability of the modified algorithm. The author's opinions on this topic are reserved for Chapter VII.

CHAPTER V FOOTNOTES

[1] Hadley, G., Non-linear and Dynamic Programming, Addison-Wesley, Reading, Massachusetts, 1964.

[2] Gonzales-Zubieta, R. H., On Some Aspects of Integer Linear Programming, Technical Report No. 16, Operations Research Center, Massachusetts Institute of Technology, June, 1965.

[3] Thompson, G. L., "The Stopped Simplex Method: I Basic Theory of Mixed Integer Programming; Integer Programming", Revue Francaise de Recherche Operationelle, Vol. 8, No. 31, pp. 159-182, 1964.

[4] Balinski, M. L., "Integer Programming: Methods, Uses, Computation", Management Science, Vol. 12, No. 3, November, 1965, pp. 253-309.

CHAPTER VI

ESSENTIALS OF A PRIMAL MILP ALGORITHM

This chapter takes the concepts behind the modified AILP as a base and generalizes them only slightly in order to outline the essentials of a primal MILP algorithm. The cuts that are generated in this algorithm were developed first by Gomory in his Algorithm III for the MILP problem (a dual feasible method). The approach here is exactly the same as was taken in developing the modified AILP algorithm. Therefore, the geometric interpretations and background material are dispensed with here in favor of a simple statement of the algorithm. An example problem is presented that takes the MILP through several steps of its procedure. Little computational experience with this algorithm is available and none is reported.

6.1 Relationships Among LP, AILP and MILP Problems.

Recall from Chapter I that both types of the ILP problem are formulated in the same way as an LP problem; the only difference being the case of ILP problems. Further, it should be obvious that all cutting plane algorithms approach ILP problems in the same manner. They all attempt to add new constraints to reduce the solution space. There are different types of Simplex algorithms (e.g., primal Simplex, dual Simplex), but every cutting plane method is, in some way, a modification

to one of these Simplex algorithms. They all solve the ILP problem by modifying the associated LP problem through the use of cuts.

In Chapter V it was seen that cuts from two seemingly very different algorithms (G-Y and Algorithm I) were actually rather strongly related. In the final statement of the modified AILP algorithm, cuts from Algorithm I were employed in a primal context and the specialized cuts of the G-Y algorithm were ignored completely. When one realizes that Algorithm I for the AILP problem can be considered to be a special case of Algorithm III for the MILP problem (the subset of integer-restricted variables contains the entire set of variables), the possibility of converting Algorithm III to a primal approach becomes evident. In fact the approach remains the same, the only difference lies in the way the cuts are generated, and not the way they are used.

6.2 Cuts for the Primal AILP Algorithm

The cuts that are generated in Algorithm III represent a generalization of the cuts generated in Algorithm I. The two algorithms are completely parallel except for the cut generation. Therefore, Algorithm III will not be reviewed; its cut generation procedure will be presented.

Consider a row of the tableau corresponding to some integer-constrained problem variable

$$x_i = a_{i,0} + \sum_{j=1}^{n} a_{i,j}(-t_j) \qquad (6.2.1)$$

Let:

R be a set composed of all non-basic (parametric) variables t_j $j=1,\ldots,n$

$R_+ \in R$ contains t_k for which $a_{i,k} > 0$ and t_k not integer-constrained

$R_- \in R$ contains t_k for which $a_{i,k} < 0$ and t_k not integer-constrained

$I \in R$ contains t_k for which t_k is integer-constrained

Gomory considered a tableau row (6.2.1) for which x_k is constrained to integer values and showed that feasible solutions that satisfy (6.2.1) also satisfy:

$$r' = -g_{i,0} + \sum_{j=1}^{n} (-g_{i,j})(-t_j) \qquad (6.2.2)$$

where the $g_{i,j}$ are selected to produce the deepest cut by the following rules:

$$g_{i,j} = \begin{cases} a_{i,j} & j \in R_+ \\ d_{i,0}\, a_{i,j} /(1-d_{i,0}) & , j \in R_- \\ d_{i,j} & , j \in I \text{ and } d_{i,j} < d_{i,0} \\ d_{i,0}(1-d_{i,j})/(1-d_{i,0}) & , j \in I \text{ and } d_{i,j} \geq d_{i,0} \end{cases}$$

The cut (6.2.2), though designed for a dual feasible method, can be used at any stage of a Simplex method in the same way as cuts from Algorithm I were employed in Chapter V. However, it should be noted that r' in (6.2.2)

is not an integer-constrained variable. If the parametric variable t_k is a non-basic proxy for an integer-constrained problem variable; t_k is also integer-constrained. If a Simplex pivot step is performed in tableau i in column k on a constraint s_i and the slack s_i is not integer-constrained, then t_k in tableau i+1 will become a non-basic proxy for s_i and as such will no longer be integer-constrained. Thus it will be necessary to keep track of the Simplex pivot steps in order to decide to which subset of R (R_+, R_-, I) a particular parametric variable belongs.

6.3 Statement of the Primal MILP Algorithm

Attention is recalled to the geometric interpretation of the Simplex pivot steps. This interpretation will not be repeated here. However, notice that the concept of stopping a Simplex method at some stage and cutting back to a (ILP) feasible solution is just as valid in the MILP case as it was seen to be in the AILP case. Therefore, the statement of the primal MILP algorithm exactly parallels the statement of the AILP algorithm. (Assume a starting basic feasible solution to the MILP problem.)

A. Select a column k for pivoting by the normal Simplex pivot selection criteria. Pivot in column k. GO TO B.

B. If the resulting tableau produces values of the problem variables that satisfy the integrality constraints, GO TO A. Otherwise, GO TO C.

C. Select some row of the tableau that represents an integer-constrained variable x_i where x_i does not have an integer value in this tableau. From this row generate the cut (6.2.2). Pivot on this cut in column k. Erase the cut. GO TO B.

This algorithm obviously is intended to terminate in Step A when the normal Simplex pivot selection criteria indicate either an optimal or an unbounded solution.

Though the procedure described above parallels the procedure of the modified AILP algorithm presented in Chapter V, it has one somewhat weaker point. When a Simplex pivot step transitions from a previously attained (MILP) feasible solution to one that is not (MILP) feasible, no precise statement can be made about the maximum number of cuts that will be needed to regain a (MILP) feasible solution. However, it is known that this will be accomplished in a finite number of iterations because Gomory proved a related supposition in his proof of convergence for Algorithm III. Without resorting to his detailed proof, the proposition may be supported by noting that column k (the pivot column) becomes lexico-positive as a result of the Simplex pivot step that causes the transition. Every successive pivot step that is performed in column k on a cut (6.2.2) produces a lexico-smaller solution and leaves column k lexico-positive. Thus the series of cuts produces a series of lexico-decreasing solutions that is bounded from below by the previous (MILP) feasible solution (since the cuts

specifically cannot exclude this solution). Gomory showed that a sequence of lexico-decreasing solutions similar to the one described here will ultimately produce the desired (MILP) feasible solution.

6.4 An Example Problem

Since this chapter has provided only a sketchy outline of the essentials of the primal MILP algorithm, and since the author's experience with this technique is very limited, the example problem presented below will illustrate the details of this algorithm. The problem that is under consideration is the same as the one used throughout this paper to demonstrate the AILP algorithms (though the objective function has been changed). However, the variable x_2 is now allowed to take on any non-negative value:

$$\begin{aligned} \max \quad & 5x_1 + 2x_2 \\ & -x_1 + 2x_2 \leq 5 \\ & x_1 + x_2 \leq 5 \\ & x_1, x_2 \geq 0 \ , \ x_1 \text{ integer} \end{aligned}$$

Figure 6.4.1 shows the initial tableau and the tableau resulting from a normal Simplex pivot in column 2. (The most negative column was not chosen in this case because this allows the MILP algorithm to be demonstrated.) Notice that x_1 has an integer value (0) so that this solution

is (MILP) feasible. The first tableau of Figure 6.4.2 is the result of another normal Simplex pivot step. However, notice that the integer restrictions on x_1 are violated. Therefore, a cut (6.2.2) is generated from the constraint row corresponding to x_1. Pivoting on this cut restores (MILP) feasibility and in this case with a better value of the objective function than was previously available. The fact that x_2 also assumes an integer value has no bearing whatsoever on the algorithm, since it is allowed to take on <u>any</u> non-negative value.

		$-t_1$	$-t_2$
$z =$	0	-5	-2
$s_1 =$	5	-1	2°
$s_2 =$	5	1	1
$x_1 =$	0	-1	0
x_2	0	0	-1

		$-t_1$	$-t_2$
$z =$	5	-6	1
$s_1 =$	0	0	-1
$s_2 =$	5/2	3/2	-1/2
$x_1 =$	0	-1	0
$x_2 =$	5/2	-1/2	1/2

Figure 6.4.1

		$-t_1$	$-t_2$
$z =$	15	4	-1
$s_1 =$	0	0	-1
$s_2 =$	0	-1	0
$x_1 =$	5/3	2/3°	-1/3
$x_2 =$	10/3	1/3	1/3
$r' =$	-2/3*	-2/3	-2/3

		$-t_1$	$-t_2$
$z =$	11	6	-5
$s_1 =$	0	0	-1
$s_2 =$	1	-3/2	1
$x_1 =$	1	1	-1
$x_2 =$	3	1/2	0
$r' =$	0	-1	0

Figure 6.4.2

CHAPTER VII

SUMMARY, CONCLUSIONS, AND SUGGESTED EXTENSIONS

7.1 Summary

The work reported here was composed of two distinct, though related, parts. The first of these was confined to a presentation of the G-Y algorithm with an accompanying discussion of the manner in which this algorithm has been coded into a problem-oriented language. Some details of the data structure and data management policies employed in this POL were discussed. Complete program documentation was included as an appendix to this report. Finally, detailed instructions on the use of this POL were given through complete documentation of the commands and their uses, an illustrative example problem encompassing all of the OPAILP's capabilities, and several random tips on some practices that should be avoided while using the system.

The second part of this paper became involved in the topic of primal ILP algorithms in general. Some geometric interpretations accompanied an explanation of a specific problem that leads the G-Y algorithm to experience unnecessarily slow convergence rates, namely the generation of redundant cuts. The first step in an attempt to show that these problems could be overcome was an illustration of how cuts taken from Gomory's Algorithm I could be employed in a primal

feasible context to produce an AILP algorithm that was equivalent to the G-Y algorithm. This equivalent algorithm demonstrated a capability of working with a completely non-integer tableau while producing primal feasible integer solutions to the AILP problem. The concept of a primal feasible AILP algorithm was altered slightly to include the case where integer solutions could be guaranteed to recur at various intervals in a modified Simplex algorithm. The cuts of Algorithm I were employed in a special way to produce a modified primal AILP algorithm. It was established that this algorithm could be guaranteed to produce primal feasible integer solutions at well defined intervals in the solution process. The performance of this algorithm was compared to various other AILP algorithms on a limited number of test problems. The results of these tests were deemed to be inconclusive.

Finally, the concepts of the modified AILP algorithm were extended to produce an outline of the essentials for the first primal feasible algorithm for the MILP case. The procedures of this algorithm were illustrated through an example problem, though no other computational results were offered.

7.2 Conclusions

No attempt has been made to compare the computational efficiency of the computer programs developed here to other existing packaged programs for the solution of AILP problems. The reason for this is that no other programs based on the G-Y algorithm are publically

available. As a result, comparisons among the various programs would be highly algorithm-dependent and could not be construed to measure the relative efficiency of the coding involved.

Along these same lines the author has specifically avoided giving estimates of the number of G-Y iterations that can be expected for problems of various sizes. While such estimates may be properly offered as rough guidelines in terms of LP problems, no such estimates are valid in terms of AILP problems. Factors other than problem size are far more relevant to the prediction of the number of iterations to be expected, namely the relative primeness of the coefficients in the various constraint rows.

As a result of the above-mentioned considerations, there remains no basis for judgements about the relative efficiency of the coding employed to develop the POL reported in the first part of this paper. The timing tests and solution attempts on a wide variety of problems that would provide the information needed to make these judgements have been subjugated to the author's more active interest in the work reported in the second part of this paper. Time constraints have not permitted the two topics to be pursued to the same extent and the former has suffered as a result. Therefore, OPAILP should be considered an experimental component of OPTECH until further computational experience can be gained.

It was reported in Chapter V that performance tests with the modified algorithm were too limited to be conclusive. On some of these

problems the modified algorithm was impressively more efficient than all other algorithms tested. However, it should be pointed out that the performance of each of the existing algorithms has always proved to be strongly influenced by the type of AILP problem that was being tested (i.e., some algorithms are very efficient on cutting stock problems and inefficient on covering problems, etc.). The author believes that this characteristic will be common to the modified algorithm. There are undoubtedly many problems that will yield more readily to the G-Y approach than to the modified algorithm. This is so because the modified algorithm was designed to cope with exactly those problems that are troublesome for the G-Y algorithm. For the problems on which G-Y is efficient, the modified algorithm will probably be relatively inefficient because cuts cannot be generated in the modified algorithm more often than in every other tableau. If the G-Y cuts on a particular problem happen to be very efficient the modified algorithm will suffer by comparison. In any event, many more tests are required before the modified algorithm can be placed in its proper prospective with regard to the other AILP algorithms. In the absence of supporting computational evidence, the author believes that the modified algorithm derives some importance from its instructional value. Its development bridges the gap between the G-Y primal algorithm and Gomory's dual feasible Algorithm I and shows that the seemingly different cuts employed in these two algorithms are, in fact, very strongly related and in some case may be equivalent.

The outline of the essentials of a primal MILP algorithm that was presented here may seem to be only an obvious extension of the modified algorithm. In fact this is true, but the author feels that this may prove to be by far the most important topic discussed here. MILP formulations abound in all areas of management and engineering, but the complexity of these problems is often directly proportional to their economic and social importance. Too often the proper formulation of these problems has been abandoned in favor of an approach that is computationally more feasible, though less precise. Therefore, the importance of developing a computational feasible MILP algorithm cannot be overemphasized. If that algorithm could be primal feasible as well, it would be all the more valuable. The author does not claim that the MILP algorithm outlined here represents the answer to this problem. However, the need for an efficient MILP algorithm is too pressing to allow any possibility to go untested.

7.3 Suggested Extensions

Work on every area encompassed by the current research has left several questions unanswered. These questions point to the obvious need for further work. However, the author feels that continuation of this work can be organized in a particular way to provide maximum return.

First the OPAILP component of OPTECH should be thoroughly tested in order to insure that the coding is precisely the best for the

situation. This testing should be performed on a variety of types of AILP probelms; specifically problems that have already been used to test other AILP algorithms. This will allow for a determination of the types of problems on which G-Y is to be preferred over some other algorithm.

Secondly an attempt should be made to interface the LP routines and the AILP routines of OPTECH to allow for the interaction necessary to solve a MILP problem via Benders' algorithm. OPAILP has been incorporated into OPTECH in such a way as to allow the requirements for data transfer through the subsystem's COMMON area.

The need for computational experience with both the modified AILP algorithm and the MILP algorithm is obvious. Investigations into the proper pivot selection and cut generation strategies for these algorithms should be more extensive. Testing of the MILP algorithm on (0,1) decision variable problems is specifically recommended. Finally, both of these algorithms lack a proof of finite convergence. Serious attempts to provide these proofs would more than likely provide the proper pivot selection and cut generation strategies as a by-product.

BIBLIOGRAPHY

Balinski, M. L., "Integer Programming: Methods, Uses, Computation", Management Science, Vol. 12, No. 3, November, 1965.

Benders, J. F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol. 4 (1962).

Ben-Israel and Charnes, A., "On Some Problems of Diophantine Programming", Cahiers de Centre d'Etudes de Recherche Operationelle, 1962.

Dantzig, G. B., "Discrete Variable Extremum Problems", Operations Research, Vol. 5 (1957).

_____, Linear Programming and Extensions, Princeton University Press, Princeton, N. J., 1963

Gomory, R. E., "An Algorithm for Integer Solutions to Linear Programs", Princeton-IBM Mathematics Research Project, Technical Report No. 1, November 17, 1958.

_____, "All-Integer Integer Programming Algorithm", IBM Research Center, Research Report RC-189, January 29, 1960.

_____, "An Algorithm for the Mixed Integer Problem", RM-2597 Rand Corporation, July 7, 1960.

Gonzales-Zubieta, R. H., On Some Aspects of Integer Linear Programming, Technical Report No. 16, Operations Research Center, Massachusetts Institute of Technology, June 1965.

Hadley, G., Non-linear and Dynamic Programming, Addison-Wesley, Reading, Massachusetts, 1964.

Jordan, J. C. (ed.), ICES: Programmers Reference Manual, Department of Civil Engineering, Massachusetts Institute of Technology, October, 1967.

Ochoa-Rosso, F., et. al., OPTECH User's Manual, Department of Civil Engineering, Massachusetts Institute of Technology, October 1967.

Sussman, J. M., Primary Memory Management in ICES: An Engineering Oriented Computer System, School of Engineering, Research Report R67-68, Massachusetts Institute of Technology, November 1967.

Thompson, G. L., "The Stopped Simplex Method: I. Basic Theory of Mixed Integer Programming; Integer Programming", Revue Francaise de Recherche Operationelle, Vol. 8, No. 31, 1964.

Young, R. D., "A Primal (all-integer) Integer Programming Algorithm", Journal of the National Bureau of Standards-B, Mathematics and Mathematical Physics, Vol. 69B, No. 3, July-September, 1965.

APPENDIX A

PROGRAM DOCUMENTATION

This Appendix contains ICETRAN listings of all of the programs that make up the CDB's for OPAILP. Each listing accompanied by an explanation of what the program does and how it fits into the sequencing of the algorithm. These explanations are contained on the side of the paper that faces the listing itself to facilitate an easy view of the program that is being discussed.

Documentation of Subroutine OPMANX

OPMANX does the following things:

1. Sets up alphanumeric data to guide the input of the tableau,L,G,S000, , OBJ, etc. All stored in LES

2. Computes the storage space required for the problem and stores this in LES(6)

3. Defines the approiate dynamic arrays to the sizes called for by data from the AILP command. NCL=no. of cols NRO=no. of rows of the problem.

4. Finally sets a flag that will let the RIGHT SIDE command be processed.

ICETRAN LISTING OF SUBROUTINE OPMANX

```
      SUBROUTINE OPMANX
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      LES(1)=1087816401
      LES(2)=1077952723
      LES(3)=1077952711
      LES(4)=1077952576
      LES(5)=-487526160
      LES(6)=4*((NCL*(NCL+NRO+6))+(3*NRO)+7)
      DEFINE IROF,NCL+NRO+1,STEP=5
      DEFINE NAMR,NCL+NRO+1,STEP=5
      DEFINE NAMC,NCL+1,STEP=1
      DEFINE ICLF,NCL+1,STEP=1
      DEFINE IAM,NCL+1,POINTER,STEP=1
      IPT=NCL+1
      DO 20 JJX=1,IPT
      ICLF(JJX)=0
   20 DEFINE IAM(JJX),NCL+NRO+1,STEP=5
      IPZ=-97
      RETURN
      END
```

Documentation of subroutine OPRHSI

OPRHSI is entered when the RIGHT SIDE command is encountered. It does the following things:

1.Checks for proper command sequencing.

2.Reads a sequence of cards noting the type of constraint (in IROF),the name of the constraint (in NAMR), and stores the coefficient value in the first column of the tableau(IAM).

3.The row names are used later in OPTAB to guide in placing the matrix coefficients in the proper place.

ICETRAN LISTING OF SUBROUTINE OPRHSI

```
      SUBROUTINE OPRHSI
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      IF(PZ+97)21,22,21
   21 WRITE(6,950)
      WRITE(6,102)
      CALL EXIT
   22 IRW=2
    1 READ(5,100) ITRM,ISLK,NRT,IAT
      IF(ITRM-LES(4))2,3,2
    2 IAM(1,1)=0
      IROF(1)=0
      IRW=IRW-1
      NAMR(1)=LES(1)
      IPZ=-99
      RETURN
    3 IF(KZZ+3)23,24,23
   24 WRITE(6,100) ITRM,ISLK,NRT,IAT
   23 NAMR(IRW)=NRT
      IAM(1,IRW)=IAT
      IF(ISLK-LES(2))5,4,5
    4 IROF(IRW)=1
      GO TO 10
    5 IF(ISLK-LES(3))7,6,7
    6 IROF(IRW)=0
      IAM(1,IRW)=IAM(1,IRW)*(-1)
      GO TO 10
    7 IF(ISLK-LES(4))9,8,9
    8 IROF(IRW)=-1
   10 IRW=IRW+1
      GO TO 1
    9 WRITE(6,100)ITRM,ISLK,NRT,IAT
      WRITE(6,101) ISLK
      WRITE(6,102)
      CALL EXIT
  100 FORMAT(3A4,2X,I6)
  101 FORMAT(//,6X,'IN ABOVE CARD-',A4,'IS ILLEGAL.')
  950 FORMAT(//,6X,'COMMANDS OUT OF SEQUENCE.')
  102 FORMAT(//,6X,'JOB FLUSHED')
      END
```

Documentation of subroutine OPTAB

Control enters OPTAB when the TABLEAU command is encountered. It does the following things:

1. It reads a sequence of data cards terminated by a non-blank field 1-4.

2. It searches the list of row names that was input in the OPRHSI subroutine. It assumes that the coefficients of any column are input together.

3. It names the columns according to the user supplied names.

4. It changes greater than or equal to inequalities to less than or equal to.

5. Once the sequence has ended, OPTAB fills out the rest of the tableau with the -I matrix representing the problem variables.

6. Various constants are set, and control is returned to the command processor.

ICETRAN LISTING OF SUBROUTINE OPTAB

```
      SUBROUTINE OPTAB
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,TE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      IF(IPZ+99)40,41,40
   40 WRITE(6,950)
      WRITE(6,102)
      CALL EXIT
   41 KT=0
      WRITE(6,105)
      KIT=2
    1 READ(5,100) ITRM,NCCK,NRCK,ITEL
      IF(ITRM-LES(4))2,3,2
    2 JAX=IRW+KIT-1
      IS=IRW+1
      DO 21 MO=1,KIT
      DO 58 KOP=IS,JAX
   58 IAM(MO,KOP)=0
      IF(MO-1)59,21,59
   59 MI=MO+IRW-1
      NAMR(MI)=NAMC(MO)
      IROF(MI)=0
      IAM(MO,MI)=-1
   21 RELEASE IAM(MO),LOW
      IE=MI
      IPZ=0
      KPIV=0
      NAMC(1)=LES(1)
      IRW=MI
      RELEASE NAMR,LOW
      RELEASE NAMC,LOW
      LEC=49
      RETURN
    3 IF(LR+2)42,43,42
   43 WRITE(6,100) ITRM,NCCK,NRCK,ITEL
   42 IF(NCCK)5,4,5
    4 WRITE(6,100) ITRM,NCCK,NRCK,ITEL
      WRITE(6,101)
      WRITE(6,102)
      CALL EXIT
    5 IF(KT)6,10,6
    6 IF(NAMC(KIT)-NCCK)7,12,7
   12 IF(NRCK)14,13,14
   13 WRITE(6,100)ITRM,NCCK,NRCK,ITEL
```

ICETRAN LISTING CONTINUED-OPTAB

```
      WRITE(6,103)
      WRITE(6,102)
      CALL EXIT
   14 DO 15 JXX=1,IRW
      IF(NAMR(JXX)-NRCK)15,17,15
   17 IF(IROF(JXX))18,19,18
   19 ITEL=ITEL*(-1)
   18 IAM(KIT,JXX)=ITEL
      GO TO 1
   15 CONTINUE
      WRITE(6,100) ITRM,NCCK,NRCK,ITEL
      WRITE(6,104) NRCK
      WRITE(6,102)
      CALL EXIT
    7 RELEASE IAM(KIT),LOW
      KIT=KIT+1
   10 NAMC(KIT)=NCCK
      KT=1
      GO TO 14
  100 FORMAT(3A4,2X,I6
  101 FORMAT(//,6X,'IN ABOVE CARD, COL. NAME IS REQ.')
  102 FORMAT(//,6X,'JOB FLUSHED')
  103 FORMAT(//,6X,'IN ABOVE CARD, ROW NAME IS REQ.')
  104 FORMAT(//,6X,'NO MATCH FOR-',A4,'-IN ABOVE CARD.')
  105 FORMAT(//)
  950 FORMAT(//,6X,'COMMANDS OUT OF SEQUENCE')
      END
```

Documentation of subroutine OPCHOZ

Control enters OPCHOZ when the ITERATE command is encountered. It does the following things:

1. It checks to see if there are any equations in the formulation.

2. If there are equations, OPCHOZ locates their row subscripts one at the time(LR). Control is passed to OPEQEL to have the equation taken care of.

3. When there ore no more equations in the tableau, control is passed to OPPSL1 where the solution process is continued until termination.

ICETRAN LISTING OF SUBROUTINE OPCHOZ

```
      SUBROUTINE OPCHOZ
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      IF(IPZ)40,41,40
   40 WRITE(6,950)
      WRITE(6,102)
      CALL EXIT
   41 KL=2
      KZ=KIT
      GO TO 4
    5 CALL OPEQEL
    4 IF(KL-IRW)9,9,10
    9 DO 2 NZ=KL,IRW
      IF(IROF(NZ))1,2,2
    1 LR=NZ
      GO TO 5
    2 CONTINUE
   10 RELEASE IROF,LOW
      WRITE(6,103) KPIV
      CALL OPPSL1
      RETURN
  102 FORMAT(//.6X,'JOB FLUSHED')
  103 FORMAT(//,6X,'FEASIBLE ITERATION',2X,I4)
  950 FORMAT(//,6X,'COMMANDS OUT OF SEQUENCE.')
      END
```

Documentation of subroutine OPEQEL

Control enters OPEQEL from OPCHOZ with a row subscript of an equation in LR. OPEQEL does the following:

1. Finds the smallest coefficient in the row LR(absolute value).

2. It puts the column subscript of the coefficient in LC.

3. Calls the computation routine OPCMP1 to have a pivot performed on IAM(LC,LR).

4. When the equation has been satisfied and there is but one coefficient in the row, this column is shifted to the last column and is no longer a part of the problem.

ICETRAN LISTING OF SUBROUTINE OPEQEL

```
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
   99 KPOZ=-1
      ILO=32000
      DO 1 JX=2,KIT
      IF(IAM(JX,LR))5,1,2
    5 ITST=IAM(JX,LR)*(-1)
      GO TO 6
    2 ITST=IAM(JX,LR)
    6 IF(ITST-ILO)3,4,4
    3 ILO=ITST
      LC=JX
    4 KPOZ=KPOZ+1
    1 RELEASE IAM(JX),LOW
      IF(KPOZ)15,8,11
    8 IF(IAM(1,LR))13,10,13
   10 IROF(LR)=1
      KL=LR+1
      SWITCH(IAM(LC),IAM(KIT))
      RELEASE IAM(LC),LOW
      RELEASE IAM(KIT),LOW
      KIT=KIT-1
      RETURN
   13 IPIT=IAM(1,LR)/IAM(LC,LR)
      IF(IAM(1,LR)-(IPIT*(IAM(LC,LR))))17,11,17
   17 NOR=NAMR(LR)
      WRITE(6,101) NOR
   77 WRITE(6,102)
   16 CALL EXIT
   11 CALL OPCMPL
      GO TO 99
   15 NOR=NAMR(LR)
      WRITE(6,200) NOR
      GO TO 77
  101 FORMAT(//,6X,'NO INTEGER SOLUTION EXISTS TO EQUATION',A4,'.')
  102 FORMAT(//,6X,JOB FLUSHED')
  200 FORMAT(//,6X,INVALID EQUATION, ALL COEFFICIENTS ARE ZERO.',/)
      END
```

Documentation of subroutine OPPSL1

Control enters OPPSL1 when there are no longer any equations in the tableau. There may or may not be another infeasibility. OPPSL1 does the following things:

1. It treats Phase I and Phase II exactly alike. It will find an infeasible row and make it a temporary objective function. It also considers for pivoting only those rows that are currently feasible.

2. It will reject a zero cut as long as there are other negative columns to be examined. If it must take a zero cut, it will take a row for which $a_{i,0} \neq 0$ where possible.

3. It also keeps a subproblem of columns that have previously generated zero cuts. (see Gonzales convergence heuristics)

4. Control continues to pass between OPPSL1 and OPCMP1 until the problem is finished through optimality or through an overrun of the iteration limit.

5. The current objective function is always stored in ICLF in order to avoid rowwise referencing through the tableau.

ICETRAN LISTING OF SUBROUTINE OPPSL1

```
      SUBROUTINE OPPSL1
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50,KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      DIMENSION IOPT(1)
      EQUIVALENCE (QQDUB(1),IOPT(1))
  102 LK=0
      LOOK=-1
    1 KZZ=1
  103 I2=(IFIND(IAM,1)+QQICOM)/4
      KZ=0
  101 KL=2
      LS=0
  104 LF=0
      LR=0
    2 IF(LK-1)3,4,4
    3 DO 6 JX=2,IRW
      I22=I2+JX
      IF(IOPT(I22))5,6,6
    5 LK=JX
      GO TO 4
    6 CONTINUE
      LK=1
      GO TO 200
    4 IF(LOOK-LK)200,201,200
  200 LOOK=LK
      DO 202 JO=1,KIT
      ICLF(JO)=IAM(JO,LK)
  202 RELEASE IAM(JO),LOW
  201 DO 7 JX=KL,KIT
      KI=JX
      IF(ICLF(JX))8,7,7
    7 CONTINUE
      IF(KL=2)46,96,46
   96 IF(LK-1)99,98,99
   99 WRITE(6,111)
      WRITE(6,105)
      CALL EXIT
   98 WRITE(6,112)
      CALL OPAUT1
      RETURN
    8 I1=(IFIND(IAM,KI)+QQICOM)/4
      IF(LS)88,89,88
   89 LC=KI
```

ICETRAN LISTING CONTINUED-OPPSL1

```
      LS=1
   88 DO 12 JX=2,IRW
      IF(JX-LK)13,12,13
   13 I11=I1+JX
      I22=I2+JX
      IF(IOPT(I22))12,15,15
   15 IF(IOPT(I11))12,12,16
   16 LF=LF+1
      IF(LF-1)9,10,9
   10 LO=IOPT(I22)/IOPT(I11)
      LR=JX
      GO TO 12
    9 ITST=IOPT(I22)/IOPT(I11)
      IF(LO-ITST)12,17,18
   18 LO=ITST
      LR=JX
      GO TO 12
   17 ITT=IOPT(I22)-(IOPT(I11)*ITST)
      ITO=IAM(1,LR)-(IAM(KI,LR)*LO)
      IF(ITO-ITT)22,19,12
  IF(ITO+ITT)20,12,20
   22 LO=ITST
      LR=JX
   20 IF(KZ)12,21,12
   21 LC=KI
      KZ=1
   12 CONTINUE
      IF(LF)30,31,30
   31 IF(LK-1)32,35,32
   35 WRITE(6,110)
      WRITE(6,105)
      CALL EXIT
   32 LC=KI
      LR=LK
      CALL OPCMP1
      GO TO 102
   30 IF(LS-1)42,49,42
   42 IF(LC-KI)41,49,41
   49 LS=LR
   41 IF(LO)43,44,43
   44 IF(KI-KIT)45,46,46
   46 LR=LS
      IF(LC-KZZ)47,47,48
   48 KZZ=KZZ+1
      SWITCH(IAM(LC),IAM(KZZ))
```

ICETRAN LISTING CONTINUED-OPPSL1

```
      ITE=ICLF(LC)
      ICLF(LC)=ICLF(KZZ)
      ICLF(KZZ)=ITE
      LC=KZZ
   47 CALL OPCMP1
      GO TO 103
   45 KL=KI+1
      RELEASE IAM(KI),LOW
      GO TO 104
   43 LC=KI
      CALL OPCMP1
      IF(LK-1)50,1,50
   50 IF(IAM(1,LK))1,102,102
  105 FORMAT(//,'JOB FLUSHED')
  110 FORMAT(//,6X,'THIS PROBLEM IS UNBOUNDED.')
  111 FORMAT(//,6X,'THIS PROBLEM HAS NO FEASIBLE SOLUTION')
  112 FORMAT(//,6X,'OPTIMALITY HAS BEEN PROVED.')
      END
```

ICETRAN LISTING OF SUBROUTINE OPCMP1

```
      SUBROUTINE OPCMP1
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),#(P),P(P),ICLF(P),KM(50,KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      DIMENSION IOPT(1)
      EQUIVALENCE (QQDUB(1),IOPT(1))
      IF(IAM(LC,LR((5,4,4
    5 IDIV=-IAM(LC,LR)
      GO TO 98
    4 IDIV=IAM(LC,LR)
   98 DO 1 KX=1,KIT
      IF(KX-LC)2,1,2
    2 IF(IAM(KX,LR))3,15,7
    3 KPO=(IAM(KX,LR)-IDIV+1)/IDIV
      GO TO 8
    7 KPO=IAM(KX,LR)/IDIV
      IF(KPO)8,15,8
    8 CALL ALOCAT (IAM,KX)
      CALL ALOCAT (IAM,LC)
      I2=(IFIND(IAM,KX)+QQICOM)/4
      I1=(IFIND(IAM,LC)+QQICOM)/4
      DO 9 J0=1,IRW
      I22=I2+J0
      I11=I1+J0
    9 IOPT(I22)=IOPT(I22)-(IOPT(I11)*KPO)
      ICLF(KX)=ICLF(KX)-ICLF(LC)*KPO)
    6 IF(LES(6)-16000,1,1,96
   96 RELEASE IAM(KX),LOW
      GO TO 1
   15 IF(KX-1)6,16,6
   16 IPZ=IPZ+1
      GO TO 6
    1 CONTINUE
      IF(IAM(LC,LR))11,11,10
   10 DO 12 KX=1,IRW
      I11=I1+KX
   12 IOPT(I11)=IOPT(I11)*(-1)
      ICLF(LC)=ICLF(LC)*(-1)
   11 KPIV=KPIV+1
      RELEASE IAM(LC),LOW
      IF(KPIV-IPIV)97,99,97
   99 WRITE(6,110)
      WRITE(6,111)
      WRITE(6,112)
```

ICETRAN LISTING CONTINUED-OPCMP1

```
      CALL OPAUT1
      CALL EXIT
   97 RETURN
  110 FORMAT(//,6X,'USER ITERATION LIMIT EXCEEDED.')
  111 FORMAT(//,6X,'CURRENT SOLUTION FOLLOWS.')
  112 FORMAT(//,6X,'THEN JOB TERMINATED.')
      END
```

Documentation of subroutine OPCMP1

Control enters OPCMP1 when an updating operation is to be performed. The subscripts of the pivot element are LC,LR. These arguments are passed through either by OPPSL1 or OPEQEL. The subroutine does the following:

1. It identifies the cut generating element.

2. The entire cut is never really appended to the tableau. Each column is updated by the pivot column in succession. The cut coefficient for each column is generated just before the column is to be updated.

3. Updating operations are straightforward. Adding or subtracting the pivot column multiplied by a scalar is all that is required to update a column.

Documentation of subroutine OPAUT1

Control enters OPAUT1 either when optimality has been proved or when the user's iteration limit has been exceeded. It merely writes out the current solution to the problem. It also names the rows in order of the way they were input; S001,S002, etc. These are the names of the slack variables associated with each constraint.

ICETRAN LISTING OF SUBROUTINE OPAUT1

```
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,IE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      DIMENSION IOPT(1)
      EQUIVALENCE (QQDUB(1),IOPT(1))
      WRITE(6,101) KPIV
      WRITE(6,103) IPZ
      IF(LEC+3)24,23,24
   24 IRWO=IS-1
      DO 25 JAG=2,IRWO
      LES(5)=LES(5)+1
   25 NAMR(JAG)=LES(5)
      GO TO 26
   23 LES(5)=LES(5)+1
      NAMR(IRW)=LES(5)
   26 I1=(IFIND(IAM,1)+QQICOM)/4
      DO 5 JX=1,IRW
      JJZ=JX+I1
      NRT=NAMR(JX)
      IZZ=IOPT(JJZ)
    5 WRITE(6,102) NRT,IZZ
  102 FORMAT(/,6X,A4,'=',I6)
  103 FORMAT(//,6X,'STATIONARY ITERATIONS',2X,I4)
  101 FORMAT(//,6X,'TOTAL ITERATIONS',2X,I4)
      KZZ=-10
      RETURN
      END
```

Documentation of subroutine OPADRW

Control enters OPADRW when the INSERT ROW command is encountered. It takes a row in the representation of the original problem statement and updates it to the current tableau representation. This is don by multiplying the row by the tableau matrix that was previously put in by OPTAB as -I. When the new row has been expressed in the new tableau representation, control is returned to the OPCHOZ subroutine to see if the addition of this new row has changed the optimal solution to this problem. If it has, then the whole problem is reoptimized.

ICETRAN LISTING OF SUBROUTINE OPADRW

```
      SUBROUTINE OPADRW
      COMMON IAM(P),B(P),NAMR(P),A(P),X(P),NAMC(P),IROF(P)
      COMMON Y(P),E(P),P(P),ICLF(P),KM(50),KBCD(9),KP(10)
      COMMON Z(15),T(6),KIT,IRW,LR,LK,LC,KPIV,KZZ,IS,IPZ,KZ
      COMMON NCAS(10),KL,TE,NCL,NRO,LES(10),IPIV,LEC
      DYNAMIC ARRAY IAM,NAMR,NAMC,IROF,ICLF
      DIMENSION IOPT(1)
      EQUIVALENCE (QQDUB(1),IOPT(1))
      IF(KZZ+10) 40,41,40
   40 WRITE(6,101)
   95 WRITE(6,104)
      CALL EXIT
   41 ICT=0
      IOP=NCL+2
      IEE=IRW+1
      IF(LES(2)-LR)35,39,35
   39 IIT=1
      IROF(IEE)=1
      GO TO 38
   35 IF(LES(3)-LR)36,37,36
   37 IIT=-1
      IROF(IEE)=0

      GO TO 38
   36 IROF(IEE)=-1
      IIT=1
   38 RELEASE IROF,LOW
      IZO=IOP-1
      DO 2 KX=1,IOP
      ICT=ICT+1
      READ(5,100)ITRM,NCCK,NRCK,ITEL
      IF(LC+1)7,9,7
    9 WRITE(6,100) ITRM,NCCK,NRCK,ITEL
    7 IF(ITRM-LES(4))3,613
  613 DO 62 KTR=1,IZO
      IF(NAMC(KTR)-NCCK)62,63,62
   63 ICLF(KTR)=ITEL*IIT
      GO TO 2
   62 CONTINUE
      WRITE(6,100) ITRM,NCCK,NRCK,ITEL
      WRITE(6,105) NCCK
      GO TO 95
    2 CONTINUE
      WRITE(6,102)
      GO TO 95
    3 IF(ICT-IOP)5,4,5
```

ICETRAN LISTING CONTINUED-OPADRW

```
     5 WRITE(6,103)
       GO TO 95
     4 DO 50 JX=1,IZO
       IF(JX-1)44,45,44
    45 IAM(JX,IEE)=ICLF(JX)
       GO TO 46
    44 IAM(JX,IEE)=0
    46 CALL ALOCAT (IAM,JX)
       I1=(IFIND(IAM,JX)+QQICOM)/4
       I33=I1+IEE
       IOT=1
       DO 47 KX=IS,IE
       IOT=IOT+1
       I11=I1+KX
    47 IOPT(I33)=IOPT(I33)-(IOPT(I11)*ICLF(IOT))
       RELEASE IAM(JX),LOW
    50 CONTINUE
       RELEASE ICLF,LOW
       KPIV=0
       IPZ=0
       IRW=IEE
       LEC=-3
       CALL OPCHOZ
       RETURN
   100 FORMAT (3A4,2X,I6)
   101 FORMAT(//,6X,'INSERT ROW COMMAND OUT OF SEQUENCE.')
   102 FORMAT(//,6X,'TOO MANY COEFF. IN INSERT ROW COMMAND.')
   103 FORMAT(//,6X,'TOO FEW COEFF. IN INSERT ROW COMMAND.')
   104 FORMAT(//,6X,'JOB FLUSHED.')
   105 FORMAT(//,6X,'NO MATCH FOUND FOR-',A4,'-IN ABOVE CARD')
       END
```